

# APPRENTISSAGE DU LANGAGE VB.NET

Serge Tahé - ISTIA - Université d'Angers  
Mars 2004

# Introduction

VB.NET est un langage récent. Il a été disponible en versions beta depuis l'année 2000 avant d'être officiellement disponible en février 2002 en même temps que la plate-forme .NET de Microsoft à laquelle il est lié. VB.NET ne peut fonctionner qu'avec cet environnement d'exécution, environnement disponible pour le moment que sur les machines Windows NT, 2000 et XP.

Avec la plate-forme .NET, trois nouveaux langages sont apparus : C#, VB.NET, JSCRIPT.NET. C# est largement une « copie » de Java. VB.NET et JSCRIPT.NET sont des extensions de Visual basic et Jscript pour la plate-forme .NET. Celle-ci rend disponible aux programmes qui s'exécutent en son sein un ensemble très important de classes, classes très proches de celles que l'on trouve au sein des machines virtuelles Java. En première approximation, on peut dire que la plate-forme .NET est un environnement d'exécution analogue à celui d'une machine virtuelle Java. On peut noter cependant deux différences importantes :

- la plate-forme .NET ne s'exécute que sur les machines Windows alors que Java s'exécute sur différents OS (windows, unix, macintosh).
- la plate-forme .NET permet l'exécution de programmes écrits en différents langages. Il suffit que le compilateur de ceux-ci sache produire du code IL (Intermediate Language), code exécuté par la machine virtuelle .NET. Toutes les classes de .NET sont disponibles aux langages compatibles .NET ce qui tend à gommer les différences entre langages dans la mesure où les programmes utilisent largement ces classes. Le choix d'un langage .NET devient affaire de goût plus que de performances.

De la même façon que Java ne peut être ignoré, la plate-forme .NET ne peut l'être, à la fois à cause du parc très important de machines windows installées et de l'effort fait par Microsoft pour la promouvoir et l'imposer. Il semble que C# soit un bon choix pour démarrer avec .NET, notamment pour les programmeurs Java, tellement ces deux langages sont proches. Ensuite on pourra passer aisément de C# à VB.NET ou à un autre langage .NET. La syntaxe changera mais les classes .NET resteront les mêmes. Contrairement aux apparences, le passage de VB à VB.NET est difficile. VB n'est pas un langage orienté objets alors que VB.NET l'est complètement. Le programmeur VB va donc être confronté à des concepts qu'il ne maîtrise pas. Il paraît plus simple d'affronter ceux-ci en oubliant ce qu'on sait de VB. Aussi, nous ne faisons que peu référence à VB dans la suite.

Ce document n'est pas un cours exhaustif. Il est destiné à des gens connaissant déjà la programmation et qui veulent découvrir VB.NET. Il reprend la structure du document "Introduction au langage C#" du même auteur afin de faciliter la comparaison entre les deux langages. En fait, j'ai utilisé ici des traducteurs automatiques de C# vers VB.NET. Sans être totalement parfaits, ces traducteurs font néanmoins 80 à 100% du travail selon les cas. On se rappellera donc, en lisant le code des programmes VB.NET qui vont suivre, que celui-ci a été d'abord généré par une machine puis remanié par moi-même si c'était nécessaire. On y rencontre ainsi des "tournures" de programmation qu'on n'aurait peut-être pas utilisé soi-même.

Les livres suivants m'ont aidé :

- Professional C# programming, Editions Wrox
- C# et .NET, Gérard Leblanc, Editions Eyrolles

Ce sont deux excellents ouvrages dont je conseille la lecture. La traduction des programmes C# en VB.NET a été obtenue par un traducteur disponible à l'url (mars 2004) <http://authors.aspalliance.com/alldotnet/examples/translate.aspx>. Tout le reste a été obtenu avec la documentation de Visual studio.NET.

Serge Tahé, mars 2004

## **1. LES BASES DU LANGAGE VB.NET.....7**

<b>1.1</b>	<b>INTRODUCTION.....</b>	<b>7</b>
<b>1.2</b>	<b>LES DONNÉES DE VB.NET.....</b>	<b>7</b>
1.2.1	LES TYPES DE DONNÉES PRÉDÉFINIS.....	7
1.2.2	NOTATION DES DONNÉES LITTÉRALES.....	8
1.2.3	DÉCLARATION DES DONNÉES.....	9
1.2.3.1	Rôle des déclarations.....	9
1.2.3.2	Déclaration des constantes.....	9
1.2.3.3	Déclaration des variables.....	9
1.2.4	LES CONVERSIONS ENTRE NOMBRES ET CHAÎNES DE CARACTÈRES.....	9
1.2.5	LES TABLEAUX DE DONNÉES.....	11
<b>1.3</b>	<b>LES INSTRUCTIONS ÉLÉMENTAIRES DE VB.NET.....</b>	<b>13</b>
1.3.1	ÉCRITURE SUR ÉCRAN.....	14
1.3.2	LECTURE DE DONNÉES TAPÉES AU CLAVIER.....	14
1.3.3	EXEMPLE D'ENTRÉES-SORTIES.....	14
1.3.4	REDIRECTION DES E/S.....	15
1.3.5	AFFECTATION DE LA VALEUR D'UNE EXPRESSION À UNE VARIABLE.....	16
1.3.5.1	Liste des opérateurs.....	16
1.3.5.2	Expression arithmétique.....	16
1.3.5.3	Priorités dans l'évaluation des expressions arithmétiques.....	18
1.3.5.4	Expressions relationnelles.....	18
1.3.5.5	Expressions booléennes.....	19
1.3.5.6	Traitement de bits.....	19
1.3.5.7	Opérateur associé à une affectation.....	20
1.3.5.8	Priorité générale des opérateurs.....	20
1.3.5.9	Les conversions de type.....	20
<b>1.4</b>	<b>LES INSTRUCTIONS DE CONTRÔLE DU DÉROULEMENT DU PROGRAMME.....</b>	<b>22</b>
1.4.1	ARRÊT.....	22
1.4.2	STRUCTURE DE CHOIX SIMPLE.....	22
1.4.3	STRUCTURE DE CAS.....	23
1.4.4	STRUCTURE DE RÉPÉTITION.....	23
1.4.4.1	Nombre de répétitions connu.....	23
1.4.4.2	Nombre de répétitions inconnu.....	24
1.4.4.3	Instructions de gestion de boucle.....	25
<b>1.5</b>	<b>LA STRUCTURE D'UN PROGRAMME VB.NET.....</b>	<b>25</b>
<b>1.6</b>	<b>COMPILATION ET EXÉCUTION D'UN PROGRAMME VB.NET.....</b>	<b>27</b>
<b>1.7</b>	<b>L'EXEMPLE IMPOTS.....</b>	<b>28</b>
<b>1.8</b>	<b>ARGUMENTS DU PROGRAMME PRINCIPAL.....</b>	<b>30</b>
<b>1.9</b>	<b>LES ÉNUMÉRATIONS.....</b>	<b>30</b>
<b>1.10</b>	<b>LA GESTION DES EXCEPTIONS.....</b>	<b>32</b>
<b>1.11</b>	<b>PASSAGE DE PARAMÈTRES À UNE FONCTION.....</b>	<b>34</b>
1.11.1	PASSAGE PAR VALEUR.....	34
1.11.2	PASSAGE PAR RÉFÉRENCE.....	35

## **2. CLASSES, STUCTURES, INTERFACES.....36**

<b>2.1</b>	<b>L' OBJET PAR L'EXEMPLE.....</b>	<b>36</b>
2.1.1	GÉNÉRALITÉS.....	36
2.1.2	DÉFINITION DE LA CLASSE PERSONNE.....	36
2.1.3	LA MÉTHODE INITIALISE.....	37
2.1.4	L'OPÉRATEUR NEW.....	37
2.1.5	LE MOT CLÉ ME.....	38
2.1.6	UN PROGRAMME DE TEST.....	38
2.1.7	UTILISER UN FICHIER DE CLASSES COMPILÉES (ASSEMBLY).....	39
2.1.8	UNE AUTRE MÉTHODE INITIALISE.....	40
2.1.9	CONSTRUCTEURS DE LA CLASSE PERSONNE.....	40
2.1.10	LES RÉFÉRENCES D'OBJETS.....	42
2.1.11	LES OBJETS TEMPORAIRES.....	43
2.1.12	MÉTHODES DE LECTURE ET D'ÉCRITURE DES ATTRIBUTS PRIVÉS.....	43
2.1.13	LES PROPRIÉTÉS.....	44
2.1.14	LES MÉTHODES ET ATTRIBUTS DE CLASSE.....	46

<a href="#">2.1.15</a>	PASSAGE D'UN OBJET À UNE FONCTION.....	48
<a href="#">2.1.16</a>	UN TABLEAU DE PERSONNES.....	49
<a href="#">2.2</a>	<b>L'HÉRITAGE PAR L'EXEMPLE.....</b>	<b>49</b>
<a href="#">2.2.1</a>	GÉNÉRALITÉS.....	49
<a href="#">2.2.2</a>	CONSTRUCTION D'UN OBJET ENSEIGNANT.....	51
<a href="#">2.2.3</a>	SURCHARGE D'UNE MÉTHODE OU D'UNE PROPRIÉTÉ.....	53
<a href="#">2.2.4</a>	LE POLYMORPHISME.....	54
<a href="#">2.2.5</a>	REDÉFINITION ET POLYMORPHISME.....	55
<a href="#">2.3</a>	<b>DÉFINIR UN INDEXEUR POUR UNE CLASSE.....</b>	<b>57</b>
<a href="#">2.4</a>	<b>LES STRUCTURES.....</b>	<b>63</b>
<a href="#">2.5</a>	<b>LES INTERFACES.....</b>	<b>66</b>
<a href="#">2.6</a>	<b>LES ESPACES DE NOMS.....</b>	<b>70</b>
<a href="#">2.7</a>	<b>L'EXEMPLE IMPOTS.....</b>	<b>72</b>

### **3. CLASSES .NET D'USAGE COURANT.....75**

<a href="#">3.1</a>	<b>CHERCHER DE L'AIDE AVEC SDK.NET.....</b>	<b>75</b>
<a href="#">3.1.1</a>	WINCV.....	75
<a href="#">3.2</a>	<b>CHERCHER DE L'AIDE SUR LES CLASSES AVEC VS.NET.....</b>	<b>78</b>
<a href="#">3.2.1</a>	OPTION AIDE.....	78
<a href="#">3.2.2</a>	AIDE/INDEX.....	80
<a href="#">3.3</a>	<b>LA CLASSE STRING.....</b>	<b>81</b>
<a href="#">3.4</a>	<b>LA CLASSE ARRAY.....</b>	<b>83</b>
<a href="#">3.5</a>	<b>LA CLASSE ARRAYLIST.....</b>	<b>85</b>
<a href="#">3.6</a>	<b>LA CLASSE HASHTABLE.....</b>	<b>87</b>
<a href="#">3.7</a>	<b>LA CLASSE STREAMREADER.....</b>	<b>91</b>
<a href="#">3.8</a>	<b>LA CLASSE STREAMWRITER.....</b>	<b>92</b>
<a href="#">3.9</a>	<b>LA CLASSE REGEX.....</b>	<b>93</b>
<a href="#">3.9.1</a>	VÉRIFIER QU'UNE CHAÎNE CORRESPOND À UN MODÈLE DONNÉ.....	95
<a href="#">3.9.2</a>	TROUVER TOUS LES ÉLÉMENTS D'UNE CHAÎNE CORRESPONDANT À UN MODÈLE.....	96
<a href="#">3.9.3</a>	RÉCUPÉRER DES PARTIES D'UN MODÈLE.....	97
<a href="#">3.9.4</a>	UN PROGRAMME D'APPRENTISSAGE.....	98
<a href="#">3.9.5</a>	LA MÉTHODE SPLIT.....	99
<a href="#">3.10</a>	<b>LES CLASSES BINARYREADER ET BINARYWRITER.....</b>	<b>100</b>

### **4. INTERFACES GRAPHIQUES AVEC VB.NET ET VS.NET..... 104**

<a href="#">4.1</a>	<b>LES BASES DES INTERFACES GRAPHIQUES.....</b>	<b>104</b>
<a href="#">4.1.1</a>	UNE FENÊTRE SIMPLE.....	104
<a href="#">4.1.2</a>	UN FORMULAIRE AVEC BOUTON.....	105
<a href="#">4.2</a>	<b>CONSTRUIRE UNE INTERFACE GRAPHIQUE AVEC VISUAL STUDIO.NET.....</b>	<b>108</b>
<a href="#">4.2.1</a>	CRÉATION INITIALE DU PROJET.....	108
<a href="#">4.2.2</a>	LES FENÊTRE DE L'INTERFACE DE VS.NET.....	110
<a href="#">4.2.3</a>	EXÉCUTION D'UN PROJET.....	111
<a href="#">4.2.4</a>	LE CODE GÉNÉRÉ PAR VS.NET.....	112
<a href="#">4.2.5</a>	COMPILATION DANS UNE FENÊTRE DOS.....	113
<a href="#">4.2.6</a>	GESTION DES ÉVÉNEMENTS.....	114
<a href="#">4.2.7</a>	CONCLUSION.....	114
<a href="#">4.3</a>	<b>FENÊTRE AVEC CHAMP DE SAISIE, BOUTON ET LIBELLÉ.....</b>	<b>115</b>
<a href="#">4.3.1</a>	CONCEPTION GRAPHIQUE.....	115
<a href="#">4.3.2</a>	GESTION DES ÉVÉNEMENTS D'UN FORMULAIRE.....	118
<a href="#">4.3.3</a>	UNE AUTRE MÉTHODE POUR GÉRER LES ÉVÉNEMENTS.....	120
<a href="#">4.3.4</a>	CONCLUSION.....	121
<a href="#">4.4</a>	<b>QUELQUES COMPOSANTS UTILES.....</b>	<b>121</b>
<a href="#">4.4.1</a>	FORMULAIRE FORM.....	121
<a href="#">4.4.2</a>	ÉTIQUETTES LABEL ET BOÎTES DE SAISIE TEXTBOX.....	122
<a href="#">4.4.3</a>	LISTES DÉROULANTES COMBOBOX.....	124
<a href="#">4.4.4</a>	COMPOSANT LISTBOX.....	125
<a href="#">4.4.5</a>	CASES À COCHER CHECKBOX, BOUTONS RADIO BUTTONRADIO.....	128
<a href="#">4.4.6</a>	VARIATEURS SCROLLBAR.....	128
<a href="#">4.5</a>	<b>ÉVÉNEMENTS SOURIS.....</b>	<b>130</b>
<a href="#">4.6</a>	<b>CRÉER UNE FENÊTRE AVEC MENU.....</b>	<b>133</b>
<a href="#">4.7</a>	<b>COMPOSANTS NON VISUELS.....</b>	<b>136</b>

<a href="#">4.7.1</a>	BOÎTES DE DIALOGUE OPENFILEDIALOG ET SAVEFILEDIALOG.....	137
<a href="#">4.7.2</a>	BOÎTES DE DIALOGUE FONTCOLOR ET COLORDIALOG.....	141
<a href="#">4.7.3</a>	TIMER.....	143
<a href="#">4.8</a>	L'EXEMPLE IMPOTS.....	145
<b><a href="#">5. GESTION D'ÉVÉNEMENTS.....</a></b>		<b>149</b>
<a href="#">5.1</a>	OBJETS DELEGATE.....	149
<a href="#">5.2</a>	GESTION D'ÉVÉNEMENTS.....	150
<a href="#">5.2.1</a>	DÉCLARATION D'UN ÉVÉNEMENT.....	150
<a href="#">5.2.2</a>	DÉFINIR LES GESTIONNAIRES D'UN ÉVÉNEMENT.....	150
<a href="#">5.2.3</a>	DÉCLENCHER UN ÉVÉNEMENT.....	150
<a href="#">5.2.4</a>	UN EXEMPLE.....	151
<b><a href="#">6. ACCÈS AUX BASES DE DONNÉES.....</a></b>		<b>155</b>
<a href="#">6.1</a>	GÉNÉRALITÉS.....	155
<a href="#">6.2</a>	LES DEUX MODES D'EXPLOITATION D'UNE SOURCE DE DONNÉES.....	156
<a href="#">6.3</a>	ACCÈS AUX DONNÉES EN MODE CONNECTÉ.....	157
<a href="#">6.3.1</a>	LES BASES DE DONNÉES DE L'EXEMPLE.....	157
<a href="#">6.3.2</a>	UTILISATION D'UN PILOTE ODBC.....	161
<a href="#">6.3.2.1</a>	La phase de connexion.....	163
<a href="#">6.3.2.2</a>	Émettre des requêtes SQL.....	163
<a href="#">6.3.2.3</a>	Exploitation du résultat d'une requête SELECT.....	164
<a href="#">6.3.2.4</a>	Libération des ressources.....	165
<a href="#">6.3.3</a>	UTILISATION D'UN PILOTE OLE DB.....	165
<a href="#">6.3.4</a>	MISE À JOUR D'UNE TABLE.....	167
<a href="#">6.3.5</a>	IMPOTS.....	170
<a href="#">6.4</a>	ACCÈS AUX DONNÉES EN MODE DÉCONNECTÉ.....	174
<b><a href="#">7. LES THREADS D'EXÉCUTION.....</a></b>		<b>175</b>
<a href="#">7.1</a>	INTRODUCTION.....	175
<a href="#">7.2</a>	CRÉATION DE THREADS D'EXÉCUTION.....	176
<a href="#">7.3</a>	INTÉRÊT DES THREADS.....	178
<a href="#">7.4</a>	ACCÈS À DES RESSOURCES PARTAGÉES.....	179
<a href="#">7.5</a>	ACCÈS EXCLUSIF À UNE RESSOURCE PARTAGÉE.....	181
<a href="#">7.6</a>	SYNCHRONISATION PAR ÉVÉNEMENTS.....	183
<b><a href="#">8. PROGRAMMATION TCP-IP.....</a></b>		<b>187</b>
<a href="#">8.1</a>	GÉNÉRALITÉS.....	187
<a href="#">8.1.1</a>	LES PROTOCOLES DE L'INTERNET.....	187
<a href="#">8.1.2</a>	LE MODÈLE OSI.....	187
<a href="#">8.1.3</a>	LE MODÈLE TCP/IP.....	188
<a href="#">8.1.4</a>	FONCTIONNEMENT DES PROTOCOLES DE L'INTERNET.....	190
<a href="#">8.1.5</a>	L'ADRESSAGE DANS L'INTERNET.....	191
<a href="#">8.1.5.1</a>	Les classes d'adresses IP.....	192
<a href="#">8.1.5.2</a>	Les protocoles de conversion Adresse Internet <--> Adresse physique.....	193
<a href="#">8.1.6</a>	LA COUCHE RÉSEAU DITE COUCHE IP DE L'INTERNET.....	193
<a href="#">8.1.6.1</a>	Le routage.....	194
<a href="#">8.1.6.2</a>	Messages d'erreur et de contrôle.....	194
<a href="#">8.1.7</a>	LA COUCHE TRANSPORT : LES PROTOCOLES UDP ET TCP.....	195
<a href="#">8.1.7.1</a>	Le protocole UDP : User Datagram Protocol.....	195
<a href="#">8.1.7.2</a>	Le protocole TCP : Transfer Control Protocol.....	195
<a href="#">8.1.8</a>	LA COUCHE APPLICATIONS.....	195
<a href="#">8.1.9</a>	CONCLUSION.....	196
<a href="#">8.2</a>	GESTION DES ADRESSES RÉSEAU.....	197
<a href="#">8.3</a>	PROGRAMMATION TCP-IP.....	199
<a href="#">8.3.1</a>	GÉNÉRALITÉS.....	199
<a href="#">8.3.2</a>	LES CARACTÉRISTIQUES DU PROTOCOLE TCP.....	200

<a href="#">8.3.3</a>	LA RELATION CLIENT-SERVEUR.....	200
<a href="#">8.3.4</a>	ARCHITECTURE D'UN CLIENT.....	200
<a href="#">8.3.5</a>	ARCHITECTURE D'UN SERVEUR.....	201
<a href="#">8.3.6</a>	LA CLASSE TcpCLIENT.....	201
<a href="#">8.3.7</a>	LA CLASSE NETWORKSTREAM.....	201
<a href="#">8.3.8</a>	ARCHITECTURE DE BASE D'UN CLIENT INTERNET.....	202
<a href="#">8.3.9</a>	LA CLASSE TcpLISTENER.....	202
<a href="#">8.3.10</a>	ARCHITECTURE DE BASE D'UN SERVEUR INTERNET.....	203
<b><a href="#">8.4</a></b>	<b>EXEMPLES.....</b>	<b>204</b>
<a href="#">8.4.1</a>	SERVEUR D'ÉCHO.....	204
<a href="#">8.4.2</a>	UN CLIENT POUR LE SERVEUR D'ÉCHO.....	206
<a href="#">8.4.3</a>	UN CLIENT TCP GÉNÉRIQUE.....	207
<a href="#">8.4.4</a>	UN SERVEUR TCP GÉNÉRIQUE.....	213
<a href="#">8.4.5</a>	UN CLIENT WEB.....	218
<a href="#">8.4.6</a>	CLIENT WEB GÉRANT LES REDIRECTIONS.....	220
<a href="#">8.4.7</a>	SERVEUR DE CALCUL D'IMPÔTS.....	223

## **9. SERVICES WEB.....** **228**

<b><a href="#">9.1</a></b>	<b>INTRODUCTION.....</b>	<b>228</b>
<b><a href="#">9.2</a></b>	<b>LES NAVIGATEURS ET XML.....</b>	<b>228</b>
<b><a href="#">9.3</a></b>	<b>UN PREMIER SERVICE WEB.....</b>	<b>229</b>
<a href="#">9.3.1</a>	VERSION 1.....	229
<a href="#">9.3.2</a>	VERSION 2.....	235
<a href="#">9.3.3</a>	VERSION 3.....	237
<a href="#">9.3.4</a>	VERSION 4.....	238
<a href="#">9.3.5</a>	CONCLUSION.....	239
<b><a href="#">9.4</a></b>	<b>UN SERVICE WEB D'OPÉRATIONS.....</b>	<b>239</b>
<b><a href="#">9.5</a></b>	<b>UN CLIENT HTTP-POST.....</b>	<b>245</b>
<b><a href="#">9.6</a></b>	<b>UN CLIENT SOAP.....</b>	<b>253</b>
<b><a href="#">9.7</a></b>	<b>ENCAPSULATION DES ÉCHANGES CLIENT-SERVEUR.....</b>	<b>257</b>
<a href="#">9.7.1</a>	LA CLASSE D'ENCAPSULATION.....	257
<a href="#">9.7.2</a>	UN CLIENT CONSOLE.....	261
<a href="#">9.7.3</a>	UN CLIENT GRAPHIQUE WINDOWS.....	263
<b><a href="#">9.8</a></b>	<b>UN CLIENT PROXY.....</b>	<b>268</b>
<b><a href="#">9.9</a></b>	<b>CONFIGURER UN SERVICE WEB.....</b>	<b>272</b>
<b><a href="#">9.10</a></b>	<b>LE SERVICE WEB IMPOTS.....</b>	<b>274</b>
<a href="#">9.10.1</a>	LE SERVICE WEB.....	274
<a href="#">9.10.2</a>	GÉNÉRER LE PROXY DU SERVICE IMPOTS.....	280
<a href="#">9.10.3</a>	UTILISER LE PROXY AVEC UN CLIENT.....	281

## **10. A SUIVRE.....** **285**

# 1. Les bases du langage VB.NET

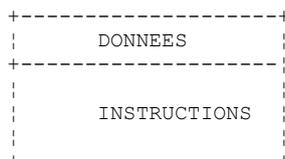
## 1.1 Introduction

Nous traitons VB.NET d'abord comme un langage de programmation classique. Nous aborderons les objets ultérieurement.

Dans un programme on trouve deux choses

- des données
- les instructions qui les manipulent

On s'efforce généralement de séparer les données des instructions :



## 1.2 Les données de VB.NET

VB.NET utilise les types de données suivants:

1. les nombres entiers, réels et décimaux
2. les caractères et chaînes de caractères
3. les booléens
4. les dates
5. les objets

### 1.2.1 Les types de données prédéfinis

Type VB	Type .NET équivalent	Taille	Plage de valeurs
Boolean	System.Boolean	2 octets	True ou False.
Byte	System.Byte	1 octet	0 à 255 (non signés).
Char	System.Char	2 octets	0 à 65 535 (non signés).
Date	System.DateTime	8 octets	00:00:00 le 1 <sup>er</sup> janvier 0001 à 23:59:59 le 31 décembre 9999.
Decimal	System.Decimal	16 octets	0 à +/-79 228 162 514 264 337 593 543 950 335 sans décimale ; 0 à +/-7,9228162514264337593543950335 avec 28 décimales ; le plus petit nombre différent de zéro étant +/-0,00000000000000000000000000000001 (+/-1E-28).
Double	System.Double	8 octets	-1,79769313486231E+308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486231E+308 pour les valeurs positives.
Integer	System.Int32	4 octets	-2 147 483 648 à 2 147 483 647.
Long	System.Int64	8 octets	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807.
Object	System.Object	4 octets	N'importe quel type peut être stocké dans une variable de type Object.
Short	System.Int16	2 octets	-32 768 à 32 767.
Single	System.Single	4 octets	-3,402823E+38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E+38 pour les valeurs positives.
String	System.String (classe)		0 à environ 2 milliards de caractères Unicode.

Dans le tableau ci-dessus, on découvre qu'il y a deux types possibles pour un entier sur 32 bits : *Integer* et *System.Int32*. Les deux types sont interchangeables. Il en est de même pour les autres types VB et leurs équivalents dans la plate-forme .NET. Voici un exemple de programme :

```

Module types
Sub Main()
    ' nombres entiers
    Dim var1 As Integer = 100
    Dim var2 As Long = 10000000000L
    Dim var3 As Byte = 100
    Dim var4 As Short = 4
    ' nombres réels
    Dim var5 As Decimal = 4.56789D
    Dim var6 As Double = 3.4
    Dim var7 As Single = -0.000103F
    ' date
    Dim var8 As Date = New Date(2003, 1, 1, 12, 8, 4)
    ' booléen
    Dim var9 As Boolean = True
    ' caractère
    Dim var10 As Char = "A"c
    ' chaîne de caractères
    Dim var11 As String = "abcde"
    ' objet
    Dim var12 As Object = New Object
    ' affichages
    Console.Out.WriteLine("var1=" + var1.ToString)
    Console.Out.WriteLine("var2=" + var2.ToString)
    Console.Out.WriteLine("var3=" + var3.ToString)
    Console.Out.WriteLine("var4=" + var4.ToString)
    Console.Out.WriteLine("var5=" + var5.ToString)
    Console.Out.WriteLine("var6=" + var6.ToString)
    Console.Out.WriteLine("var7=" + var7.ToString)
    Console.Out.WriteLine("var8=" + var8.ToString)
    Console.Out.WriteLine("var9=" + var9.ToString)
    Console.Out.WriteLine("var10=" + var10)
    Console.Out.WriteLine("var11=" + var11)
    Console.Out.WriteLine("var12=" + var12.ToString)
End Sub
End Module

```

L'exécution donne les résultats suivants :

```

var1=100
var2=10000000000
var3=100
var4=4
var5=4,56789
var6=3,4
var7=-0,000103
var8=01/01/2003 12:08:04
var9=True
var10=A
var11=abcde
var12=System.Object

```

## 1.2.2 Notation des données littérales

Integer	145, -7, &FF (hexadécimal)
Long	100000L
Double	134.789, -45E-18 (-45 10 <sup>-18</sup> )
Single	134.789F, -45E-18F (-45 10 <sup>-18</sup> )
Decimal	100000D
Char	"A"c
String	"aujourd'hui"
Boolean	true, false
date	<a href="#">New Date</a> (2003, 1, 1) pour le 01/01/2003

On notera les points suivants :

- 100000L, le L pour signifier qu'on considère le nombre comme un entier long
- 134.789F, le F pour signifier qu'on considère le nombre comme un réel simple précision
- 100000D, le D pour signifier qu'on considère le nombre comme un réel décimal

- "A"c, pour transformer la chaîne de caractères "A" en caractère 'A'
- la chaîne de caractères est entouré du caractère ". Si la chaîne doit contenir le caractère ", on double celui-ci comme dans "abcd""e" pour représenter la chaîne [abcd"e].

## 1.2.3 Déclaration des données

### 1.2.3.1 Rôle des déclarations

Un programme manipule des données caractérisées par un nom et un type. Ces données sont stockées en mémoire. Au moment de la traduction du programme, le compilateur affecte à chaque donnée un emplacement en mémoire caractérisé par une adresse et une taille. Il le fait en s'aidant des déclarations faites par le programmeur. Par ailleurs celles-ci permettent au compilateur de détecter des erreurs de programmation. Ainsi l'opération  $x=x*2$  sera déclarée erronée si  $x$  est une chaîne de caractères par exemple.

### 1.2.3.2 Déclaration des constantes

La syntaxe de déclaration d'une constante est la suivante :

**const identificateur as type=valeur**

par exemple [const PI as double=3.141592]. Pourquoi déclarer des constantes ?

1. La lecture du programme sera plus aisée si l'on a donné à la constante un nom significatif : [const taux\_tva as single=0.186F]
2. La modification du programme sera plus aisée si la "constante" vient à changer. Ainsi dans le cas précédent, si le taux de tva passe à 33%, la seule modification à faire sera de modifier l'instruction définissant sa valeur : [const taux\_tva as single=0.336F]. Si l'on avait utilisé 0.186 explicitement dans le programme, ce serait alors de nombreuses instructions qu'il faudrait modifier.

### 1.2.3.3 Déclaration des variables

Une variable est identifiée par un nom et se rapporte à un type de données. VB.NET ne fait pas la différence entre majuscules et minuscules. Ainsi les variables **FIN** et **fin** sont identiques. Les variables peuvent être initialisées lors de leur déclaration. La syntaxe de déclaration d'une ou plusieurs variables est :

**dim variable1,variable2,...,variablen as identificateur\_de\_type**

où *identificateur\_de\_type* est un type prédéfini ou bien un type défini par le programmeur.

## 1.2.4 Les conversions entre nombres et chaînes de caractères

nombre -> chaîne	<i>nombre.ToString</i> ou <i>"" &amp; nombre</i> ou <i>CType(nombre,String)</i>
objet -> chaîne	<i>objet.ToString</i>
chaîne -> Integer	<i>Integer.Parse(chaîne)</i> ou <i>Int32.Parse</i>
chaîne -> Long	<i>Long.Parse(chaîne)</i> ou <i>Int64.Parse</i>
chaîne -> Double	<i>Double.Parse(chaîne)</i>
chaîne -> Single	<i>Single.Parse(chaîne)</i>

La conversion d'une chaîne vers un nombre peut échouer si la chaîne ne représente pas un nombre valide. Il y a alors génération d'une erreur fatale appelée **exception** en VB.NET. Cette erreur peut être gérée par la clause *try/catch* suivante :

```
try
    appel de la fonction susceptible de générer l'exception
catch e as Exception
    traiter l'exception e
end try
instruction suivante
```

Si la fonction ne génère pas d'exception, on passe alors à **instruction suivante**, sinon on passe dans le corps de la clause *catch* puis à **instruction suivante**. Nous reviendrons ultérieurement sur la gestion des exceptions. Voici un programme présentant les principales techniques de conversion entre nombres et chaînes de caractères. Dans cet exemple la fonction **affiche** écrit à l'écran la valeur de son paramètre. Ainsi *affiche(S)* écrit la valeur de **S** à l'écran.

```
' directives
Option Strict On

' espaces de noms importés
Imports System
```

```

' le module de test
Module Module1

Sub Main()
' procédure principale
' données locales
Dim S As String
Const i As Integer = 10
Const l As Long = 100000
Const f As Single = 45.78F
Dim d As Double = -14.98

' nombre --> chaîne
affiche(CType(i, String))
affiche(CType(l, String))
affiche(CType(f, String))
affiche(CType(d, String))

'boolean --> chaîne
Const b As Boolean = False
affiche(b.ToString)

' chaîne --> int
Dim i1 As Integer = Integer.Parse("10")
affiche(i1.ToString)
Try
    i1 = Integer.Parse("10.67")
    affiche(i1.ToString)
Catch e As Exception
    affiche("Erreur [10.67] : " + e.Message)
End Try

' chaîne --> long
Dim l1 As Long = Long.Parse("100")
affiche("" + l1.ToString)
Try
    l1 = Long.Parse("10.675")
    affiche("" & l1)
Catch e As Exception
    affiche("Erreur [10.675] : " + e.Message)
End Try

' chaîne --> double
Dim d1 As Double = Double.Parse("100,87")
affiche(d1.ToString)
Try
    d1 = Double.Parse("abcd")
    affiche("" & d1)
Catch e As Exception
    affiche("Erreur [abcd] : " + e.Message)
End Try

' chaîne --> single
Dim f1 As Single = Single.Parse("100,87")
affiche(f1.ToString)
Try
    d1 = Single.Parse("abcd")
    affiche(f1.ToString)
Catch e As Exception
    affiche("Erreur [abcd] : " + e.Message)
End Try
End Sub

' affiche
Public Sub affiche(ByVal S As String)
    Console.Out.WriteLine("S=" + S)
End Sub
End Module

```

Les résultats obtenus sont les suivants :

```

S=10
S=100000
S=45,78
S=-14,98
S=False
S=10
S=Erreur [10.67] : Le format de la chaîne d'entrée est incorrect.
S=100
S=Erreur [10.675] : Le format de la chaîne d'entrée est incorrect.

```

```
S=100,87
S=Erreur [abcd] : Le format de la chaîne d'entrée est incorrect.
S=100,87
S=Erreur [abcd] : Le format de la chaîne d'entrée est incorrect.
```

On remarquera que les nombres réels sous forme de chaîne de caractères doivent utiliser la virgule et non le point décimal. Ainsi on écrira `Dim d As Double = -14.98` mais `Dim d1 As Double = Double.Parse("100,87")`

## 1.2.5 Les tableaux de données

Un tableau VB.NET est un objet permettant de rassembler sous un même identificateur des données de même type. Sa déclaration est la suivante :

**Dim Tableau(n) as type** ou **Dim Tableau() as type=New type(n) {}**

où **n** est l'indice du dernier élément de tableau. La syntaxe *Tableau(i)* désigne la donnée n° *i* où *i* appartient à l'intervalle  $[0, n]$ . Toute référence à la donnée *Tableau(i)* où *i* n'appartient pas à l'intervalle  $[0, n]$  provoquera une exception. Un tableau peut être initialisé en même temps que déclaré. Dans ce cas, on n'a pas besoin d'indiquer le n° du dernier élément.

```
Dim entiers() As Integer = {0, 10, 20, 30}
```

Les tableaux ont une propriété **Length** qui est le nombre d'éléments du tableau. Voici un programme exemple :

```
Module tab0
Sub Main()
' un premier tableau
Dim tab0(5) As Integer
For i As Integer = 0 To UBound(tab0)
    tab0(i) = i
Next
For i As Integer = 0 To UBound(tab0)
    Console.Out.WriteLine("tab0(" + i.ToString + ")=" + tab0(i).ToString)
Next

' un second tableau
Dim tab1() As Integer = New Integer(5) {}
For i As Integer = 0 To tab1.Length - 1
    tab1(i) = i * 10
Next
For i As Integer = 0 To tab1.Length - 1
    Console.Out.WriteLine("tab1(" + i.ToString + ")=" + tab1(i).ToString)
Next
End Sub
End Module
```

et son exécution :

```
tab0(0)=0
tab0(1)=1
tab0(2)=2
tab0(3)=3
tab0(4)=4
tab0(5)=5
tab1(0)=0
tab1(1)=10
tab1(2)=20
tab1(3)=30
tab1(4)=40
tab1(5)=50
```

Un tableau à deux dimensions pourra être déclaré comme suit :

**Dim Tableau(n,m) as Type** ou **Dim Tableau(,) as Type=New Type(n,m) {}**

où  $n+1$  est le nombre de lignes,  $m+1$  le nombre de colonnes. La syntaxe *Tableau(i,j)* désigne l'élément *j* de la ligne *i* de *Tableau*. Le tableau à deux dimensions peut lui aussi être initialisé en même temps qu'il est déclaré :

```
Dim réels(,) As Double = {{0.5, 1.7}, {8.4, -6}}
```

Le nombre d'éléments dans chacune des dimensions peut être obtenue par la méthode **GetLenth(i)** où  $i=0$  représente la dimension correspondant au 1er indice,  $i=1$  la dimension correspondant au 2ième indice, ... Voici un programme d'exemple :

```
Module Module2
```

```

Sub Main()
    ' un premier tableau
    Dim tab0(2, 1) As Integer
    For i As Integer = 0 To UBound(tab0)
        For j As Integer = 0 To tab0.GetLength(1) - 1
            tab0(i, j) = i * 10 + j
        Next
    Next
    For i As Integer = 0 To UBound(tab0)
        For j As Integer = 0 To tab0.GetLength(1) - 1
            Console.Out.WriteLine("tab0(" + i.ToString + "," + j.ToString + ")=" + tab0(i, j).ToString)
        Next
    Next

    ' un second tableau
    Dim tab1(,) As Integer = New Integer(2, 1) {}
    For i As Integer = 0 To tab1.GetLength(0) - 1
        For j As Integer = 0 To tab1.GetLength(1) - 1
            tab1(i, j) = i * 100 + j
        Next
    Next
    For i As Integer = 0 To tab1.GetLength(0) - 1
        For j As Integer = 0 To tab1.GetLength(1) - 1
            Console.Out.WriteLine("tab1(" + i.ToString + "," + j.ToString + ")=" + tab1(i, j).ToString)
        Next
    Next
End Sub
End Module

```

et les résultats de son exécution :

```

tab0(0)=0
tab0(1)=1
tab0(2)=2
tab0(3)=3
tab0(4)=4
tab0(5)=5
tab1(0)=0
tab1(1)=10
tab1(2)=20
tab1(3)=30
tab1(4)=40
tab1(5)=50

```

Un tableau de tableaux est déclaré comme suit :

**Dim Tableau(n)() as Type** ou **Dim Tableau()() as Type=new Type(n)**

La déclaration ci-dessus crée un tableau de  $n+1$  lignes. Chaque élément *Tableau(i)* est une référence de tableau à une dimension. Ces tableaux ne sont pas créés lors de la déclaration ci-dessus. L'exemple ci-dessous illustre la création d'un tableau de tableaux :

```

' un tableau de tableaux
Dim noms()() As String = New String(3)() {}
' initialisation
For i = 0 To noms.Length - 1
    noms(i) = New String(i) {}
    For j = 0 To noms(i).Length - 1
        noms(i)(j) = "nom" & i & j
    Next
Next

```

Ici *noms(i)* est un tableau de  $i+1$  éléments. Comme *noms(i)* est un tableau, *noms(i).Length* est son nombre d'éléments. Voici un exemple regroupant les trois types de tableaux que nous venons de présenter :

```

' directives
Option Strict On
Option Explicit On

' imports
Imports System

' classe de test
Module test
    Sub main()
        ' un tableau à 1 dimension initialisé
        Dim entiers() As Integer = {0, 10, 20, 30}
        Dim i As Integer
    End Sub
End Module

```

```

For i = 0 To entiers.Length - 1
    Console.Out.WriteLine("entiers[" & i & "]=" & entiers(i))
Next

' un tableau à 2 dimensions initialisé
Dim réels(,) As Double = {{0.5, 1.7}, {8.4, -6}}
Dim j As Integer
For i = 0 To réels.GetLength(0) - 1
    For j = 0 To réels.GetLength(1) - 1
        Console.Out.WriteLine("réels[" & i & ", " & j & "]=" & réels(i, j))
    Next
Next

' un tableau°de tableaux
Dim noms()() As String = New String(3)() {}

' initialisation
For i = 0 To noms.Length°- 1
    noms(i) =°New String(i) {}
    For j = 0 To noms(i).Length - 1
        noms(i)(j) = "nom" & i & j
    Next
Next

' affichage
For i = 0 To noms.Length°- 1
    For j = 0 To noms(i).Length - 1
        Console.Out.WriteLine("noms[" & i & "]" & j & "]=" & noms(i)(j))
    Next
Next
End Sub
End Module

```

A l'exécution, nous obtenons les résultats suivants :

```

entiers[0]=0
entiers[1]=10
entiers[2]=20
entiers[3]=30
réels[0,0]=0,5
réels[0,1]=1,7
réels[1,0]=8,4
réels[1,1]=-6
noms[0][0]=nom00
noms[1][0]=nom10
noms[1][1]=nom11
noms[2][0]=nom20
noms[2][1]=nom21
noms[2][2]=nom22
noms[3][0]=nom30
noms[3][1]=nom31
noms[3][2]=nom32
noms[3][3]=nom33

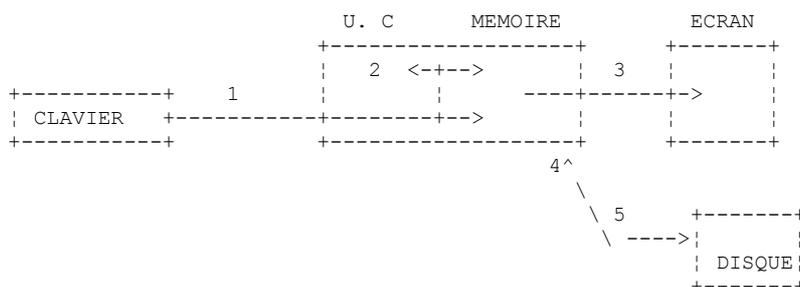
```

## 1.3 Les instructions élémentaires de VB.NET

On distingue

- 1 les instructions élémentaires exécutées par l'ordinateur.
- 2 les instructions de contrôle du déroulement du programme.

Les instructions élémentaires apparaissent clairement lorsqu'on considère la structure d'un micro-ordinateur et de ses périphériques.



1. lecture d'informations provenant du clavier
2. traitement d'informations
3. écriture d'informations à l'écran
4. lecture d'informations provenant d'un fichier disque
5. écriture d'informations dans un fichier disque

### 1.3.1 Écriture sur écran

Il existe différentes instructions d'écriture à l'écran :

```
Console.Out.WriteLine(expression)
Console.WriteLine(expression)
Console.Error.WriteLine(expression)
```

où *expression* est tout type de donnée qui puisse être converti en chaîne de caractères pour être affiché à l'écran. Dans les exemples vus jusqu'ici, nous n'avons utilisé que l'instruction *Console.Out.WriteLine(expression)*.

La classe *System.Console* donne accès aux opérations d'écriture écran (**Write**, **WriteLine**). La classe *Console* a deux propriétés **Out** et **Error** qui sont des **flux d'écriture** de type *StreamWriter* :

- *Console.WriteLine()* est équivalent à *Console.Out.WriteLine()* et écrit sur le flux **Out** associé habituellement à l'écran.
- *Console.Error.WriteLine()* écrit sur le flux **Error**, habituellement associé lui aussi l'écran.

Les flux *Out* et *Error* sont associés par défaut l'écran. Mais ils peuvent être redirigés vers des fichiers texte au moment de l'exécution du programme comme nous le verrons prochainement.

### 1.3.2 Lecture de données tapées au clavier

Le flux de données provenant du clavier est désigné par l'objet *Console.In* de type *StreamReader*. Ce type d'objets permet de lire une ligne de texte avec la méthode *ReadLine* :

```
Dim ligne As String = Console.In.ReadLine()
```

La ligne tapée au clavier est rangée dans la variable *ligne* et peut ensuite être exploitée par le programme. Le flux **In** peut être redirigé vers un fichier comme les flux **Out** et **Error**.

### 1.3.3 Exemple d'entrées-sorties

Voici un court programme d'illustration des opérations d'entrées-sorties clavier/écran :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' module
Module io1
Sub Main()
' écriture sur le flux Out
Dim obj As New Object
Console.Out.WriteLine("'" & obj.ToString())

' écriture sur le flux Error
Dim i As Integer = 10
Console.Error.WriteLine("i=" & i)

' lecture d'une ligne saisie au clavier
Console.Out.Write("Tapez une ligne : ")
Dim ligne As String = Console.In.ReadLine()
Console.Out.WriteLine("ligne=" + ligne)
End Sub
End Module
```

et les résultats de l'exécution :

```
System.Object
i=10
Tapez une ligne : ceci est un essai
ligne=ceci est un essai
```

```
Dim obj As New Object
Console.Out.WriteLine(obj.ToString)
```

ne sont là que pour montrer que n'importe quel objet peut faire l'objet d'un affichage. Nous ne chercherons pas ici à expliquer la signification de ce qui est affiché.

### 1.3.4 Redirection des E/S

Il existe sous DOS/Windows trois périphériques standard appelés :

1. périphérique d'entrée standard - désigne par défaut le clavier et porte le n° 0
2. périphérique de sortie standard - désigne par défaut l'écran et porte le n° 1
3. périphérique d'erreur standard - désigne par défaut l'écran et porte le n° 2

En VB.NET, le flux d'écriture *Console.Out* écrit sur le périphérique 1, le flux d'écriture *Console.Error* écrit sur le périphérique 2 et le flux de lecture *Console.In* lit les données provenant du périphérique 0. Lorsqu'on lance un programme dans une fenêtre Dos sous Windows, on peut fixer quels seront les périphériques 0, 1 et 2 pour le programme exécuté. Considérons la ligne de commande suivante :

```
pg arg1 arg2 .. argn
```

Derrière les arguments *argi* du programme *pg*, on peut rediriger les périphériques d'E/S standard vers des fichiers:

0<in.txt	le flux d'entrée standard n° 0 est redirigé vers le fichier <i>in.txt</i> . Dans le programme le flux <i>Console.In</i> prendra donc ses données dans le fichier <i>in.txt</i> .
1>out.txt	redirige la sortie n° 1 vers le fichier <i>out.txt</i> . Cela entraîne que dans le programme le flux <i>Console.Out</i> écrira ses données dans le fichier <i>out.txt</i>
1>>out.txt	idem, mais les données écrites sont ajoutées au contenu actuel du fichier <i>out.txt</i> .
2>error.txt	redirige la sortie n° 2 vers le fichier <i>error.txt</i> . Cela entraîne que dans le programme le flux <i>Console.Error</i> écrira ses données dans le fichier <i>error.txt</i>
2>>error.txt	idem, mais les données écrites sont ajoutées au contenu actuel du fichier <i>error.txt</i> .
1>out.txt 2>error.txt	Les périphériques 1 et 2 sont tous les deux redirigés vers des fichiers

On notera que pour rediriger les flux d'E/S du programme *pg* vers des fichiers, le programme *pg* n'a pas besoin d'être modifié. C'est l'OS qui fixe la nature des périphériques 0,1 et 2. Considérons le programme suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' redirections
Module console2
Sub Main()
    ' lecture flux In
    Dim data As String = Console.In.ReadLine()
    ' écriture flux Out
    Console.Out.WriteLine("écriture dans flux Out : " + data)
    ' écriture flux Error
    Console.Error.WriteLine("écriture dans flux Error : " + data)
End Sub
End Module
```

Compilons ce programme :

```
dos>vbc es2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573
Copyright (C) Microsoft Corporation 1987-2002. Tous droits réservés.

dos>dir
24/02/2004 15:39          416 es2.vb
11/03/2004 08:20        3 584 es2.exe
```

Faisons une première exécution:

```
dos>es2.exe
un premier test
écriture dans flux Out : un premier test
écriture dans flux Error : un premier test
```

L'exécution précédente ne redirige aucun des flux d'E/S standard **In**, **Out**, **Error**. Nos allons maintenant rediriger les trois flux. Le flux *In* sera redirigé vers un fichier *in.txt*, le flux *Out* vers le fichier *out.txt*, le flux *Error* vers le fichier *error.txt*. Cette redirection a lieu sur la ligne de commande sous la forme

```
dos>es2.exe 0<in.txt 1>out.txt 2>error.txt
```

L'exécution donne les résultats suivants :

```
dos>more in.txt
un second test

dos>es2.exe 0<in.txt 1>out.txt 2>error.txt

dos>more out.txt
écriture dans flux Out : un second test

dos>more error.txt
écriture dans flux Error : un second test
```

On voit clairement que les flux *Out* et *Error* n'écrivent pas sur les mêmes périphériques.

## 1.3.5 Affectation de la valeur d'une expression à une variable

On s'intéresse ici à l'opération **variable=expression**. L'expression peut être de type : arithmétique, relationnelle, booléenne, caractères.

### 1.3.5.1 Liste des opérateurs

Action	Élément du langage
Arithmétique	^, -, *, /, \, Mod, +, =
Assignment	=, ^=, *=, /=, \=, +=, -=, &=
Comparaison	=, <>, <, >, <=, >=, Like, Is
Concaténation	&, +
Opérations logiques/de bits	Not, And, Or, Xor, AndAlso, OrElse
Opérations diverses	AddressOf, GetType

### 1.3.5.2 Expression arithmétique

Les opérateurs des expressions arithmétiques sont les suivants :

Arithmétique	^, -, *, /, \, Mod, +, =
--------------	--------------------------

+ : addition, - : soustraction, \* : multiplication, / : division réelle, \ : quotient de division entière, Mod : reste de la division entière, ^ : élévation à la puissance. Ainsi le programme suivant :

```
' opérateurs arithmétiques
Module operateursarithmetiques
Sub Main()
    Dim i, j As Integer
    i = 4 : j = 3
    Console.Out.WriteLine(i & "/" & j & "=" & (i / j))
    Console.Out.WriteLine(i & "\" & j & "=" & (i \ j))
    Console.Out.WriteLine(i & " mod " & j & "=" & (i Mod j))

    Dim r1, r2 As Double
    r1 = 4.1 : r2 = 3.6
    Console.Out.WriteLine(r1 & "/" & r2 & "=" & (r1 / r2))
    Console.Out.WriteLine(r1 & "^2=" & (r1 ^ 2))
    Console.Out.WriteLine(Math.Cos(3))
End Sub
```

donne les résultats suivants :

```
4/3=1,33333333333333
4\3=1
4 mod 3=1
4,1/3,6=1,13888888888889
4,1^2=16,81
-0,989992496600445
```

Il existe diverses fonctions mathématiques. En voici quelques-unes :

Public Shared Function Sqrt(ByVal d As Double) As Double	racine carrée
Public Shared Function Cos(ByVal d As Double) As Double	Cosinus
Public Shared Function Sin(ByVal a As Double) As Double	Sinus
Public Shared Function Tan(ByVal a As Double) As Double	Tangente
Public Shared Function Pow(ByVal x As Double,ByVal y As Double) As Double	x à la puissance y (x>0)
Public Shared Function Exp(ByVal d As Double) As Double	Exponentielle
Overloads Public Shared Function Log( ByVal d As Double ) As Double	Logarithme népérien
Overloads Public Shared Function Abs(ByVal value As Double) As Double	valeur absolue
....	

Toutes ces fonctions sont définies dans une classe .NET appelée **Math**. Lorsqu'on les utilise, il faut les préfixer avec le nom de la classe où elles sont définies. Ainsi on écrira :

```
Dim r1, r2 As Double
r2 = Math.Sqrt(9)
r1 = Math.Cos(3)
```

La définition complète de la classe *Math* est la suivante :

E	Représente la base de logarithme naturelle spécifiée par la constante <b>e</b> .
PI	Représente le rapport de la circonférence d'un cercle à son diamètre, spécifié par la constante $\pi$ .
Abs	Surchargé. Retourne la valeur absolue d'un nombre spécifié.
Acos	Retourne l'angle dont le cosinus est le nombre spécifié.
Asin	Retourne l'angle dont le sinus est le nombre spécifié.
Atan	Retourne l'angle dont la tangente est le nombre spécifié.
Atan2	Retourne l'angle dont la tangente est le quotient de deux nombres spécifiés.
BigMul	Génère le produit intégral de deux nombres 32 bits.
Ceiling	Retourne le plus petit nombre entier supérieur ou égal au nombre spécifié.
Cos	Retourne le cosinus de l'angle spécifié.
Cosh	Retourne le cosinus hyperbolique de l'angle spécifié.
DivRem	Surchargé. Retourne le quotient de deux nombres, en passant le reste en tant que paramètre de sortie.
Exp	Retourne <b>e</b> élevé à la puissance spécifiée.
Floor	Retourne le plus grand nombre entier inférieur ou égal au nombre spécifié.
IEEERemainder	Retourne le reste de la division d'un nombre spécifié par un autre.
Log	Surchargé. Retourne le logarithme d'un nombre spécifié.
Log10	Retourne le logarithme de base 10 d'un nombre spécifié.
Max	Surchargé. Retourne le plus grand de deux nombres spécifiés.
Min	Surchargé. Retourne le plus petit de deux nombres.

Pow	Retourne un nombre spécifié élevé à la puissance spécifiée.
Round	Surchargé. Retourne le nombre le plus proche de la valeur spécifiée.
Sign	Surchargé. Retourne une valeur indiquant le signe d'un nombre.
Sin	Retourne le sinus de l'angle spécifié.
Sinh	Retourne le sinus hyperbolique de l'angle spécifié.
Sqrt	Retourne la racine carrée d'un nombre spécifié.
Tan	Retourne la tangente de l'angle spécifié.
Tanh	Retourne la tangente hyperbolique de l'angle spécifié.

Lorsqu'une fonction est déclarée "surchargée", c'est qu'elle existe pour divers type de paramètres. Par exemple, la fonction Abs(x) existe pour x de type Integer, Long, Decimal, Single, Float. Pour chacun de ces types existe une définition séparée de la fonction Abs. On dit alors qu'elle est surchargée.

### 1.3.5.3 Priorités dans l'évaluation des expressions arithmétiques

La priorité des opérateurs lors de l'évaluation d'une expression arithmétique est la suivante (du plus prioritaire au moins prioritaire) :

Catégorie	Opérateurs
Primaire	Toutes les expressions sans opérateur : fonctions, parenthèses
Élévation à la puissance	^
Négation unaire	+, -
Multiplication	*, /
Division par un entier	\
Modulo	Mod
Addition	+, -

### 1.3.5.4 Expressions relationnelles

Les opérateurs sont les suivants :

Comparaison	=, <>, <, >, <=, >=, Like, Is
-------------	-------------------------------

= : égal à, <> : différent de, < : plus petit que (strictement), > : plus grand que (strictement), <= : inférieur ou égal, >= : supérieur ou égal, Like : correspond à un modèle, Is : identité d'objets. Tous ces opérateurs ont la même priorité. Ils sont évalués de la gauche vers la droite. Le résultat d'une expression relationnelle un booléen.

**Comparaison de chaînes de caractères** : considérons le programme suivant :

```
' espaces de noms
Imports System

Module string1
Sub main()
Dim ch1 As Char = "A"c
Dim ch2 As Char = "B"c
Dim ch3 As Char = "a"c
Console.Out.WriteLine("A<B=" & (ch1 < ch2))
Console.Out.WriteLine("A<a=" & (ch1 < ch3))
Dim chat As String = "chat"
Dim chien As String = "chien"
Dim chaton As String = "chaton"
Dim chat2 As String = "CHAT"
Console.Out.WriteLine("chat<chien=" & (chat < chien))
Console.Out.WriteLine("chat<chaton=" & (chat < chaton))
Console.Out.WriteLine("chat<CHAT=" & (chat < chat2))
Console.Out.WriteLine("chaton like chat*" & ("chaton" Like "chat*"))
End Sub
End Module
```

et le résultat de son exécution :

```
A<B=True
Les bases de VB.NET
```

```
A<a=True
chat<chien=True
chat<chaton=True
chat<CHAT=False
chaton like chat*=True
```

Soient deux caractères C1 et C2. Il est possible de les comparer avec les opérateurs : <, <=, =, <>, >, >=. Ce sont alors leurs valeurs Unicode des caractères, qui sont des nombres, qui sont comparées. Selon l'ordre Unicode, on a les relations suivantes :

espace < .. < '0' < '1' < .. < '9' < .. < 'A' < 'B' < .. < 'Z' < .. < 'a' < 'b' < .. < 'z'

Les chaînes de caractères sont comparées caractère par caractère. La première inégalité rencontrée entre deux caractères induit une inégalité de même sens sur les chaînes. Avec ces explications, le lecteur est invité à étudier les résultats du programme précédent.

### 1.3.5.5 Expressions booléennes

Les opérateurs sont les suivants :

Opérations logiques/de bits | Not, And, Or, Xor, AndAlso, OrElse

Not : et logique, Or : ou logique, Not : négation, Xor : ou exclusif.  
 op1 AndAlso op2 : si op1 est faux, op2 n'est pas évalué et le résultat est faux.  
 op1 OrElse op2 : si op1 est vrai, op2 n'est pas évalué et le résultat est vrai.

La priorité de ces opérateurs entre-eux est la suivante :

NOT logique	Not
AND logique	And, AndAlso
OR logique	Or, OrElse
XOR logique	Xor

Le résultat d'une expression booléenne est un booléen.

### 1.3.5.6 Traitement de bits

On retrouve d'une part les mêmes opérateurs que les opérateurs booléens avec la même priorité. On trouve également deux opérateurs de déplacement : << et >>. Soient i et j deux entiers.

i<<n	décale i de n bits sur la gauche. Les bits entrants sont des zéros.
i>>n	décale i de n bits sur la droite. Si i est un entier signé (signed char, int, long) le bit de signe est préservé.
i & j	fait le ET logique de i et j bit à bit.
i   j	fait le OU logique de i et j bit à bit.
~i	complémente i à 1
i^j	fait le OU EXCLUSIF de i et j

Soit le programme suivant :

```
Module operationsbit
Sub main()
' manipulation de bits
Dim i As Short = &H123F
Dim k As Short = &H7123
Console.Out.WriteLine("i<<4=" & (i << 4).ToString("X"))
Console.Out.WriteLine("i>>4=" & (i >> 4).ToString("X"))
Console.Out.WriteLine("k>>4=" & (k >> 4).ToString("X"))
Console.Out.WriteLine("i and 4=" & (i And 4).ToString("X"))
Console.Out.WriteLine("i or 4 =" & (i Or 4).ToString("X"))
Console.Out.WriteLine("not i=" & (Not i).ToString("X"))
End Sub
End Module
```

Son exécution donne les résultats suivants :

```
i<<4=23F0
i>>4=123
k>>4=712
i and 4=4
```

```
i or k =123F
not i=EDC0
```

### 1.3.5.7 Opérateur associé à une affectation

Il est possible d'écrire `a+=b` qui signifie `a=a+b`. La liste des opérateurs pouvant se combiner avec l'opération d'affectation est la suivante :

combinaison d'opérateurs | ^=, \*=, /=, \=, +=, -=, &=

### 1.3.5.8 Priorité générale des opérateurs

Catégorie	Opérateurs
Primaire	Toutes les expressions sans opérateur
Élévation à la puissance	^
Négation unaire	+, -
Multiplication	*, /
Division par un entier	\
Modulo	Mod
Addition	+, -
Concaténation	&
Déplacement	<<, >>
Relationnel	=, <>, <, >, <=, >=, Like, Is, TypeOf...Is
NOT logique	Not
AND logique	And, AndAlso
OR logique	Or, OrElse
XOR logique	Xor

Lorsqu'un opérande est placé entre deux opérateurs de même priorité, l'associativité des opérateurs régit l'ordre dans lequel les opérations sont effectuées. Tous les opérateurs sont associatifs à gauche, ce qui signifie que les opérations sont exécutées de gauche à droite. La priorité et l'associativité peuvent être contrôlées à l'aide d'expressions entre parenthèses.

### 1.3.5.9 Les conversions de type

Il existe un certain nombre de fonction prédéfinies permettant de passer d'un type de données à un autre. Leur liste est la suivante :

**CBool, CByte, CChar, CDate, CDb1, CDec, CInt, CLng, CObj, CShort, CSng, CStr**

Ces fonctions acceptent comme argument une expression numérique ou une chaîne de caractères. Le type du résultat est indiqué dans le tableau suivant :

fonction	résultat	Domaine de valeurs du paramètre de la fonction
<b>CBool</b>	Boolean	Toute chaîne ou expression numérique valide.
<b>CByte</b>	Byte	0 à 255 ; les fractions sont arrondies.
<b>CChar</b>	Char	Toute expression String valide ; la valeur peut être comprise entre 0 et 65 535.
<b>CDate</b>	Date	Toute représentation valide de la date et de l'heure.
<b>CDbl</b>	Double	-1,79769313486231E+308 à -4,94065645841247E-324 pour les valeurs négatives ; 4,94065645841247E-324 à 1,79769313486231E+308 pour les valeurs positives.
<b>CDec</b>	Decimal	+/-79 228 162 514 264 337 593 543 950 335 pour les nombres sans décimales. La plage de valeurs des nombres à 28 décimales est +/-7,9228162514264337593543950335. Le plus petit nombre différent de zéro est 0,00000000000000000000000000000001.
<b>CInt</b>	Integer	-2 147 483 648 à 2 147 483 647 ; les fractions sont arrondies.

fonction	résultat	Domaine de valeurs du paramètre de la fonction
<b>CLng</b>	Long	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 ; les fractions sont arrondies.
<b>CObj</b>	Object	Toute expression valide.
<b>CShort</b>	Short	-32 768 à 32 767 ; les fractions sont arrondies.
<b>CSng</b>	Single	-3,402823E+38 à -1,401298E-45 pour les valeurs négatives ; 1,401298E-45 à 3,402823E+38 pour les valeurs positives.
<b>CStr</b>	String	Les valeurs retournées par la fonction Cstr dépendent de l'argument expression.

Voici un programme exemple :

```
Module conversion
Sub main()
    Dim var1 As Boolean = CBool("true")
    Dim var2 As Byte = CByte("100")
    Dim var3 As Char = CChar("A")
    Dim var4 As Date = CDate("30 janvier 2004")
    Dim var5 As Double = CDbl("100,45")
    Dim var6 As Decimal = CDec("1000,67")
    Dim var7 As Integer = CInt("-30")
    Dim var8 As Long = CLng("456")
    Dim var9 As Short = CShort("-14")
    Dim var10 As Single = CSng("56,78")
    Console.Out.WriteLine("var1=" & var1)
    Console.Out.WriteLine("var2=" & var2)
    Console.Out.WriteLine("var3=" & var3)
    Console.Out.WriteLine("var4=" & var4)
    Console.Out.WriteLine("var5=" & var5)
    Console.Out.WriteLine("var6=" & var6)
    Console.Out.WriteLine("var7=" & var7)
    Console.Out.WriteLine("var8=" & var8)
    Console.Out.WriteLine("var9=" & var9)
    Console.Out.WriteLine("var10=" & var10)
End Sub
End Module
```

et les résultats de son exécution :

```
var1=True
var2=100
var3=A
var4=30/01/2004
var5=100,45
var6=1000,67
var7=-30
var8=456
var9=-14
var10=56,78
```

On peut également utiliser la fonction **CType(expression, type)** comme le montre le programme suivant :

```
Module ctype1
Sub main()
    Dim var1 As Boolean = CType("true", Boolean)
    Dim var2 As Byte = CType("100", Byte)
    Dim var3 As Char = CType("A", Char)
    Dim var4 As Date = CType("30 janvier 2004", Date)
    Dim var5 As Double = CType("100,45", Double)
    Dim var6 As Decimal = CType("1000,67", Decimal)
    Dim var7 As Integer = CType("-30", Integer)
    Dim var8 As Long = CType("456", Long)
    Dim var9 As Short = CType("-14", Short)
    Dim var10 As Single = CType("56,78", Single)
    Dim var11 As String = CType("47,89", String)
    Dim var12 As String = 47.89.ToString
    Dim var13 As String = "" & 47.89
    Console.Out.WriteLine("var1=" & var1)
    Console.Out.WriteLine("var2=" & var2)
    Console.Out.WriteLine("var3=" & var3)
    Console.Out.WriteLine("var4=" & var4)
    Console.Out.WriteLine("var5=" & var5)
    Console.Out.WriteLine("var6=" & var6)
    Console.Out.WriteLine("var7=" & var7)
    Console.Out.WriteLine("var8=" & var8)
    Console.Out.WriteLine("var9=" & var9)
```

```

    Console.Out.WriteLine("var10=" & var10)
    Console.Out.WriteLine("var11=" & var11)
    Console.Out.WriteLine("var12=" & var12)
    Console.Out.WriteLine("var13=" & var13)
End Sub
End Module

```

qui donne les résultats suivants :

```

var1=True
var2=100
var3=A
var4=30/01/2004
var5=100,45
var6=1000,67
var7=-30
var8=456
var9=-14
var10=56,78
var11=47,89
var12=47,89
var13=47,89

```

## 1.4 Les instructions de contrôle du déroulement du programme

### 1.4.1 Arrêt

La méthode `Exit` définie dans la classe `Environment` permet d'arrêter l'exécution d'un programme :

```
Public Shared Sub Exit(ByVal exitCode As Integer )
```

arrête le processus en cours et rend la valeur `exitCode` au processus père. La valeur de `exitCode` peut être utilisée par celui-ci. Sous DOS, cette variable status est rendue à DOS dans la variable système **ERRORLEVEL** dont la valeur peut être testée dans un fichier batch. Sous Unix, c'est la variable `$?` qui récupère la valeur de `exitCode`.

```
Environment.Exit(0)
```

arrêtera l'exécution du programme avec une valeur d'état à 0.

### 1.4.2 Structure de choix simple

```

if condition then
    actions_alors
else
    actions_sinon
end if

```

- chaque action est sur une ligne
- la clause `else` peut être absente.

On peut imbriquer les structures de choix comme le montre l'exemple suivant :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module if1
    Sub main()
        Dim i As Integer = 10
        If i > 4 Then
            Console.Out.WriteLine(i & " est > " & 4)
        Else
            If i = 4 Then
                Console.Out.WriteLine(i & " est = " & 4)
            Else
                Console.Out.WriteLine(i & " est < " & 4)
            End If
        End If
    End Sub
End Module

```

```
End Sub
End Module
```

Le résultat obtenu :

```
10 est > 4
```

## 1.4.3 Structure de cas

La syntaxe est la suivante :

```
select case expression
case liste_valeurs1
    actions1
case liste_valeurs2
    actions2
...
case else
    actions_sinon
end select
```

- le type de [expression] doit être l'un des types suivants :

```
Boolean, Byte, Char, Date, Decimal, Double, Integer, Long, Object, Short, Single et String
```

- la clause [case else] peut être absente.
- [liste\_valeursi] sont des valeurs possibles de l'expression. [listes\_valeursi] représente une liste de conditions condition1, condition2, ..., conditionx. Si [expression] vérifie l'une des conditions, les actions derrière la clause [liste\_valeursi] sont exécutées. Les conditions peuvent prendre la forme suivante :
  - val1 to val2 : vrai si [expression] appartient au domaine [val1,val2]
  - val1 : vrai si [expression] est égal à val1
  - is > val1 : vrai si [expression] > val1. Le mot clé [is] peut être absent
  - idem avec les opérateurs =, <, <=, >, >=, <>
- seules les actions liées à la première condition vérifiée sont exécutées.

Considérons le programme suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module selectcase1
Sub main()
    Dim i As Integer = 10
    Select Case i
        Case 1 To 4, 7 To 8
            Console.Out.WriteLine("i est dans l'intervalle [1,4] ou [7,8]")
        Case Is > 12
            Console.Out.WriteLine("i est > 12")
        Case Is < 15
            Console.Out.WriteLine("i est < 15")
        Case Is < 20
            Console.Out.WriteLine("i est < 20")
    End Select
End Sub
End Module
```

Il donne les résultats suivants :

```
i est < 15
```

## 1.4.4 Structure de répétition

### 1.4.4.1 Nombre de répétitions connu

```
For counter [ As datatype ] = start To end [ Step step ]
    actions
```

```
Next [ counter ]
```

Les actions sont effectuées pour chacune des valeurs prises par la variable [counter]. Considérons le programme suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module for1
Sub main()
    Dim somme As Integer = 0
    Dim résultat As String = "somme("
    For i As Integer = 0 To 10 Step 2
        somme += i
        résultat += " " + i.ToString
    Next
    résultat += ")=" + somme.ToString
    Console.Out.WriteLine(résultat)
End Sub
End Module
```

Les résultats :

```
somme( 0 2 4 6 8 10)=30
```

Une autre structure d'itération à nombre d'itérations connu est la suivante :

```
For Each element [ As datatype ] In groupe
    [ actions ]
Next [ element ]
```

- *groupe* est une collection d'objets. La collection d'objets que nous connaissons déjà est le tableau
- *datatype* est le type des objets de la collection. Pour un tableau, ce serait le type des éléments du tableau
- *element* est une variable locale à la boucle qui va prendre successivement pour valeur, toutes les valeurs de la collection.

Ainsi le code suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module foreach1
Sub main()
    Dim amis() As String = {"paul", "hélène", "jacques", "sylvie"}
    For Each nom As String In amis
        Console.Out.WriteLine(nom)
    Next
End Sub
End Module
```

afficherait :

```
paul
hélène
jacques
sylvie
```

#### 1.4.4.2 Nombre de répétitions inconnu

Il existe de nombreuses structures en VB.NET pour ce cas.

```
Do { While | Until } condition
    [ statements ]
Loop
```

On boucle tant que la condition est vérifiée (while) ou jusqu'à ce que la condition soit vérifiée (until). La boucle peut ne jamais être exécutée.

```
Do
  [ statements ]
Loop { While | Until } condition
```

On boucle tant que la condition est vérifiée (while) ou jusqu'à ce que la condition soit vérifiée (until). La boucle est toujours exécutée au moins une fois.

```
While condition
  [ statements ]
End While
```

On boucle tant que la condition est vérifiée. La boucle peut ne jamais être exécutée. Les boucles suivantes calculent toutes la somme des 10 premiers nombres entiers.

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module boucles1
  Sub main()
    Dim i, somme As Integer
    i = 0 : somme = 0
    Do While i < 11
      somme += i
      i += 1
    Loop
    Console.Out.WriteLine("somme=" + somme.ToString)
    i = 0 : somme = 0
    Do Until i = 11
      somme += i
      i += 1
    Loop
    Console.Out.WriteLine("somme=" + somme.ToString)
    i = 0 : somme = 0
    Do
      somme += i
      i += 1
    Loop Until i = 11
    Console.Out.WriteLine("somme=" + somme.ToString)
    i = 0 : somme = 0
    Do
      somme += i
      i += 1
    Loop While i < 11
    Console.Out.WriteLine("somme=" + somme.ToString)
  End Sub
End Module
```

```
somme=55
somme=55
somme=55
somme=55
```

### 1.4.4.3 Instructions de gestion de boucle

`exit do` fait sortir d'une boucle do ... loop

`exit for` fait sortir d'une boucle for

## 1.5 La structure d'un programme VB.NET

Un programme VB.NET n'utilisant pas de classe définie par l'utilisateur ni de fonctions autres que la fonction principale *Main* pourra avoir la structure suivante :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports espace1
```

```
Imports ....
Module nomDuModule
  Sub main()
  ....
End Sub
End Module
```

- La directive [Option Explicit on] force la déclaration des variables. En VB.NET, celle-ci n'est pas obligatoire. Une variable non déclarée est alors de type **Object**.
- La directive [Option Strict on] interdit toute conversion de types de données pouvant entraîner une perte de données et toute conversion entre les types numériques et les chaînes. Il faut alors explicitement utiliser des fonctions de conversion.
- Le programme importe tous les espaces de noms dont il a besoin. Nous n'avons pas introduit encore cette notion. Nous avons, dans les programmes précédents, souvent rencontré des instructions du genre :

```
Console.Out.WriteLine(unechaine)
```

Nous aurions du écrire en fait :

```
System.Console.Out.WriteLine(unechaine)
```

où **System** est l'espace de noms contenant la classe [Console]. En important l'espace de noms [System] avec une instruction Imports, VB.NET l'explorera systématiquement lorsqu'il rencontrera une classe qu'il ne connaît pas. Il répétera la recherche avec tous les espaces de noms déclarés jusqu'à trouver la classe recherchée. On écrit alors :

```
' espaces de noms
Imports System
....
Console.Out.WriteLine(unechaine)
```

Un exemple de programme pourrait être le suivant :

```
' options
Option Explicit On
Option Strict On

'espaces de noms
Imports System

' module principal
Module main1
  Sub main()
    Console.Out.WriteLine("main1")
  End Sub
End Module
```

Le même programme peut être écrit de la façon suivante :

```
' options
Option Explicit On
Option Strict On

'espaces de noms
Imports System

' classe de test
Public Class main2
  Public Shared Sub main()
    Console.Out.WriteLine("main2")
  End Sub
End Class
```

Ici, nous utilisons le concept de classe qui sera introduit au chapitre suivant. Lorsqu'une telle classe contient une procédure statique (shared) appelée **main**, celle-ci est exécutée. Si nous introduisons cette écriture ici, c'est parce que le langage jumeau de VB.NET qu'est C# ne connaît que le concept de classe, c.a.d. que tout code exécuté appartient nécessairement à une classe. La notion de classe appartient à la programmation objet. L'imposer dans la conception de tout programme est un peu maladroit. On le voit ici dans la version 2 du programme précédent où on est amené à introduire un concept de classe et de méthode statique là où il n'y en a pas besoin. Aussi, par la suite, n'introduirons-nous le concept de classe que lorsqu'il est nécessaire. Dans les autres cas, nous utiliserons la notion de module comme dans la version 1 ci-dessus.

## 1.6 Compilation et exécution d'un programme VB.NET

La compilation d'un programme VB.NET ne nécessite que le SDK.NET. Prenons le programme suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module boucles1
Sub main()
Dim i, somme As Integer
i = 0 : somme = 0
Do While i < 11
    somme += i
    i += 1
Loop
Console.Out.WriteLine("somme=" + somme.ToString)
i = 0 : somme = 0
Do Until i = 11
    somme += i
    i += 1
Loop
Console.Out.WriteLine("somme=" + somme.ToString)
i = 0 : somme = 0
Do
    somme += i
    i += 1
Loop Until i = 11
Console.Out.WriteLine("somme=" + somme.ToString)
i = 0 : somme = 0
Do
    somme += i
    i += 1
Loop While i < 11
Console.Out.WriteLine("somme=" + somme.ToString)
End Sub
End Module
```

Supposons qu'il soit dans un fichier appelé [boucles1.vb]. Pour le compiler, nous procédons ainsi :

```
dos>dir boucles1.vb
11/03/2004 15:55                583 boucles1.vb

dos>vbc boucles1.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4
pour Microsoft (R) .NET Framework version 1.1.4322.573
Copyright (C) Microsoft Corporation 1987-2002. Tous droits réservés.

dos>dir boucles1.*
11/03/2004 16:04                601 boucles1.vb
11/03/2004 16:04                3 584 boucles1.exe
```

Le programme `vbc.exe` est le compilateur VB.NET. Ici, il était dans le PATH du DOS :

```
dos>path
PATH=E:\Program Files\Microsoft Visual Studio .NET 2003\Common7\IDE;E:\Program Files\Microsoft Visual Studio .NET 2003\VC7\BIN;E:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools;E:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\bin\prerelease;E:\Program Files\Microsoft Visual Studio .NET 2003\Common7\Tools\bin;E:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1\bin;E:\WINNT\Microsoft.NET\Framework\v1.1.4322;e:\winnt\system32;e:\winnt;
```

```
dos>dir E:\WINNT\Microsoft.NET\Framework\v1.1.4322\vbc.exe
21/02/2003 10:20                737 280 vbc.exe
```

Le compilateur [vbc] produit un fichier .exe exécutable par la machine virtuelle .NET :

```
dos>boucles1
somme=55
somme=55
somme=55
somme=55
```

## 1.7 L'exemple IMPOTS

On se propose d'écrire un programme permettant de calculer l'impôt d'un contribuable. On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer :

- on calcule le nombre de parts du salarié  $\text{nbParts} = \text{nbEnfants} / 2 + 1$  s'il n'est pas marié,  $\text{nbEnfants} / 2 + 2$  s'il est marié, où  $\text{nbEnfants}$  est son nombre d'enfants.
- s'il a au moins trois enfants, il a une demi-part de plus
- on calcule son revenu imposable  $\text{R} = 0.72 * \text{S}$  où S est son salaire annuel
- on calcule son coefficient familial  $\text{QF} = \text{R} / \text{nbParts}$
- on calcule son impôt I. Considérons le tableau suivant :

12620.0	0	0
13190	0.05	631
15640	0.1	1290.5
24740	0.15	2072.5
31810	0.2	3309.5
39970	0.25	4900
48360	0.3	6898.5
55790	0.35	9316.5
92970	0.4	12106
127860	0.45	16754.5
151250	0.50	23147.5
172040	0.55	30710
195000	0.60	39312
0	0.65	49062

Chaque ligne a 3 champs. Pour calculer l'impôt I, on recherche la première ligne où  $\text{QF} \leq \text{champ1}$ . Par exemple, si  $\text{QF} = 23000$  on trouvera la ligne

**24740 0.15 2072.5**

L'impôt I est alors égal à  $0.15 * \text{R} - 2072.5 * \text{nbParts}$ . Si QF est tel que la relation  $\text{QF} \leq \text{champ1}$  n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

**0 0.65 49062**

ce qui donne l'impôt  $I = 0.65 * \text{R} - 49062 * \text{nbParts}$ . Le programme VB.NET correspondant est le suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module impots
    ' ----- main
    Sub Main()

        ' tableaux de données nécessaires au calcul de l'impôt
        Dim Limites() As Decimal = {12620D, 13190D, 15640D, 24740D, 31810D, 39970D, 48360D, 55790D, 92970D,
127860D, 151250D, 172040D, 195000D, 0D}
        Dim CoeffN() As Decimal = {0D, 631D, 1290.5D, 2072.5D, 3309.5D, 4900D, 6898.5D, 9316.5D, 12106D,
16754.5D, 23147.5D, 30710D, 39312D, 49062D}

        ' on récupère le statut marital
        Dim OK As Boolean = False
        Dim reponse As String = Nothing
        While Not OK
            Console.Out.Write("Etes-vous marié(e) (O/N) ? ")
            reponse = Console.In.ReadLine().Trim().ToLower()
            If reponse <> "o" And reponse <> "n" Then
                Console.Error.WriteLine("Réponse incorrecte. Recommencez")
            Else
                OK = True
            End If
        End While
        Dim Marie As Boolean = reponse = "o"

        ' nombre d'enfants
        OK = False
        Dim NbEnfants As Integer = 0
        While Not OK
            Console.Out.Write("Nombre d'enfants : ")
            reponse = Console.In.ReadLine()
            Try
                NbEnfants = Integer.Parse(reponse)
```

```

    If NbEnfants >= 0 Then
        OK = True
    Else
        Console.Error.WriteLine("Réponse incorrecte. Recommencez")
    End If
End Try
Catch
    Console.Error.WriteLine("Réponse incorrecte. Recommencez")
End Try
End While
' salaire
OK = False
Dim Salaire As Integer = 0
While Not OK
    Console.Out.Write("Salaire annuel : ")
    reponse = Console.In.ReadLine()
    Try
        Salaire = Integer.Parse(reponse)
        If Salaire >= 0 Then
            OK = True
        Else
            Console.Error.WriteLine("Réponse incorrecte. Recommencez")
        End If
    Catch
        Console.Error.WriteLine("Réponse incorrecte. Recommencez")
    End Try
End While
' calcul du nombre de parts
Dim NbParts As Decimal
If Marie Then
    NbParts = CDec(NbEnfants) / 2 + 2
Else
    NbParts = CDec(NbEnfants) / 2 + 1
End If
If NbEnfants >= 3 Then
    NbParts += 0.5D
End If
' revenu imposable
Dim Revenu As Decimal
Revenu = 0.72D * Salaire

' quotient familial
Dim QF As Decimal
QF = Revenu / NbParts

' recherche de la tranche d'impôts correspondant à QF
Dim i As Integer
Dim NbTranches As Integer = Limites.Length
Limites((NbTranches - 1)) = QF
i = 0
While QF > Limites(i)
    i += 1
End While
' l'impôt
Dim impots As Integer = CInt(i * 0.05D * Revenu - CoeffN(i) * NbParts)

' on affiche le résultat
Console.Out.WriteLine(("Impôt à payer : " & impots))
End Sub
End Module

```

Le programme est compilé dans une fenêtre Dos par :

```

dos>vbc impots1.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>dir impots1.exe
24/02/2004 15:42          5 632 impots1.exe

```

La compilation produit un exécutable *impots.exe*. Il faut noter que *impots.exe* n'est pas directement exécutable par le processeur. Il contient en réalité du code intermédiaire qui n'est exécutable que sur une plate-forme .NET. Les résultats obtenus sont les suivants :

```

dos>impots1
Etes-vous marié(e) (O/N) ? o
Nombre d'enfants : 3
Salaire annuel : 200000
Impôt à payer : 16400

```

```
dos>impots1
Etes-vous marié(e) (O/N) ? n
Nombre d'enfants : 2
Salaire annuel : 200000
Impôt à payer : 33388
```

```
dos>impots1
Etes-vous marié(e) (O/N) ? w
Réponse incorrecte. Recommencez
Etes-vous marié(e) (O/N) ? q
Réponse incorrecte. Recommencez
Etes-vous marié(e) (O/N) ? o
Nombre d'enfants : q
Réponse incorrecte. Recommencez
Nombre d'enfants : 2
Salaire annuel : q
Réponse incorrecte. Recommencez
Salaire annuel : 1
Impôt à payer : 0
```

## 1.8 Arguments du programme principal

La procédure principale *Main* peut admettre comme paramètre un tableau de chaînes :

```
Sub main(ByVal args() As String)
```

Le paramètre **args** est un tableau de chaînes de caractères qui reçoit les arguments passés sur la ligne de commande lors de l'appel du programme.

- *args.Length* est le nombre d'éléments du tableau *args*
- *args(i)* est l'élément *i* du tableau

Si on lance le programme P avec la commande : `P arg0 arg1 ... argn` et si la procédure *Main* du programme P est déclarée comme suit :

```
Sub main(ByVal args() As String)
```

on aura `arg(0)="arg0"`, `arg(1)="arg1"` ... Voici un exemple :

```
' directives
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module arg
  Sub main(ByVal args() As String)
    ' nombre d'arguments
    console.out.writeline("Il y a " & args.length & " arguments")
    Dim i As Integer
    For i = 0 To args.Length - 1
      Console.Out.WriteLine("argument n° " & i & "=" & args(i))
    Next
  End Sub
End Module
```

L'exécution donne les résultats suivants :

```
dos>arg1 a b c
Il y a 3 arguments
argument n° 0=a
argument n° 1=b
argument n° 2=c
```

## 1.9 Les énumérations

Une énumération est un type de données dont le domaine de valeurs est un ensemble de constantes entières. Considérons un programme qui a à gérer des mentions à un examen. Il y en aurait cinq : *Passable*, *AssezBien*, *Bien*, *TrèsBien*, *Excellent*. On pourrait alors définir une énumération pour ces cinq constantes :

```
Enum mention
```

```

Passable
AssezBien
Bien
TrèsBien
Excellent
End Enum

```

De façon interne, ces cinq constantes sont codées par des entiers consécutifs commençant par 0 pour la première constante, 1 pour la suivante, etc... Une variable peut être déclarée comme prenant ces valeurs dans l'énumération :

```

' une variable qui prend ses valeurs dans l'énumération mention
Dim maMention As mention = mention.Passable

```

On peut comparer une variable aux différentes valeurs possibles de l'énumération :

```

' test avec valeur de l'énumération
If (maMention = mention.Passable) Then
    Console.Out.WriteLine("Peut mieux faire")
End If

```

On peut obtenir toutes les valeurs de l'énumération :

```

For Each m In mention.GetValues (maMention.GetType)
    Console.Out.WriteLine (m)
Next

```

De la même façon que le type simple *Integer* est équivalent à la structure *Int32*, le type simple *Enum* est équivalent à la structure *Enum*. Cette classe a une méthode statique *GetValues* qui permet d'obtenir toutes les valeurs d'un type énuméré que l'on passe en paramètre. Celui-ci doit être un objet de type **Type** qui est une classe d'information sur le type d'une donnée. Le type d'une variable *v* est obtenu par *v.GetType()*. Donc ici *maMention.GetType()* donne l'objet *Type* de l'énumération *mentions* et *Enum.GetValues(maMention.GetType())* la liste des valeurs de l'énumération *mentions*.

C'est ce que montre le programme suivant :

```

' directives
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Module enum2

    ' une énumération
    Enum mention
        Passable
        AssezBien
        Bien
        TrèsBien
        Excellent
    End Enum

    ' pg de test
    Sub Main()

        ' une variable qui prend ses valeurs dans l'énumération mention
        Dim maMention As mention = mention.Passable

        ' affichage valeur variable
        Console.Out.WriteLine("mention=" & maMention)

        ' test avec valeur de l'énumération
        If (maMention = mention.Passable) Then
            Console.Out.WriteLine("Peut mieux faire")
        End If

        ' liste des mentions littérales
        For Each m As mention In [Enum].GetValues (maMention.GetType)
            Console.Out.WriteLine (m.ToString)
        Next

        ' liste des mentions entières
        For Each m As Integer In [Enum].GetValues (maMention.GetType)
            Console.Out.WriteLine (m)
        Next
    End Sub
End Module

```

Les résultats d'exécution sont les suivants :

```
dos>enum2
mention=0
Peut mieux faire
Passable
AssezBien
Bien
TrèsBien
Excellent
0
1
2
3
4
```

## 1.10 La gestion des exceptions

De nombreuses fonctions VB.NET sont susceptibles de générer des exceptions, c'est à dire des erreurs. Lorsqu'une fonction est susceptible de générer une exception, le programmeur devrait la gérer dans le but d'obtenir des programmes plus résistants aux erreurs : il faut toujours éviter le "plantage" sauvage d'une application.

La gestion d'une exception se fait selon le schéma suivant :

```
try
    appel de la fonction susceptible de générer l'exception
catch e as Exception e)
    traiter l'exception e
end try
instruction suivante
```

Si la fonction ne génère pas d'exception, on passe alors à *instruction suivante*, sinon on passe dans le corps de la clause *catch* puis à *instruction suivante*. Notons les points suivants :

- **e** est un objet dérivé du type *Exception*. On peut être plus précis en utilisant des types tels que *IOException*, *SystemException*, etc... : il existe plusieurs types d'exceptions. En écrivant *catch e as Exception*, on indique qu'on veut gérer toutes les types d'exceptions. Si le code de la clause *try* est susceptible de générer plusieurs types d'exceptions, on peut vouloir être plus précis en gérant l'exception avec plusieurs clauses *catch* :

```
try
    appel de la fonction susceptible de générer l'exception
catch e as IOException
    traiter l'exception e
catch e as SystemException
    traiter l'exception e
end try
instruction suivante
```

- On peut ajouter aux clauses *try/catch*, une clause **finally** :

```
try
    appel de la fonction susceptible de générer l'exception
catch e as Exception
    traiter l'exception e
finally
    code exécuté après try ou catch
end try
instruction suivante
```

Qu'il y ait exception ou pas, le code de la clause *finally* sera toujours exécuté.

- Dans la clause *catch*, on peut ne pas vouloir utiliser l'objet *Exception* disponible. Au lieu d'écrire *catch e as Exception*, on écrit alors *catch*.
- La classe *Exception* a une propriété **Message** qui est un message détaillant l'erreur qui s'est produite. Ainsi si on veut afficher celui-ci, on écrira :

```
catch e as Exception
    Console.Error.WriteLine("L'erreur suivante s'est produite : "+e.Message);
    ...
```

```
end try
```

- La classe *Exception* a une méthode **Tostring** qui rend une chaîne de caractères indiquant le type de l'exception ainsi que la valeur de la propriété *Message*. On pourra ainsi écrire :

```
catch ex as Exception
    Console.Error.WriteLine("L'erreur suivante s'est produite : "+ex.ToString)
...
end try
```

L'exemple suivant montre une exception générée par l'utilisation d'un élément de tableau inexistant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module tabl
Sub Main()
    ' déclaration & initialisation d'un tableau
    Dim tab() As Integer = {0, 1, 2, 3}
    Dim i As Integer

    ' affichage tableau avec un for
    For i = 0 To tab.Length - 1
        Console.Out.WriteLine(("tab[" & i & "]=" & tab(i)))
    Next i

    ' affichage tableau avec un for each
    Dim élmt As Integer
    For Each élmt In tab
        Console.Out.WriteLine(élmt)
    Next élmt

    ' génération d'une exception
    Try
        tab(100) = 6
    Catch e As Exception
        Console.Error.WriteLine(("L'erreur suivante s'est produite : " & e.Message))
    End Try
End Sub
End Module
```

L'exécution du programme donne les résultats suivants :

```
dos>exception1
tab[0]=0
tab[1]=1
tab[2]=2
tab[3]=3
0
1
2
3
L'erreur suivante s'est produite : L'index se trouve en dehors des limites du tableau.
```

Voici un autre exemple où on gère l'exception provoquée par l'affectation d'une chaîne de caractères à un nombre lorsque la chaîne ne représente pas un nombre :

```
' options
Option Strict On
Option Explicit On

'imports
Imports System

Public Module console1
    Public Sub Main()
        ' On demande le nom
        System.Console.Write("Nom : ")

        ' lecture réponse
        Dim nom As String = System.Console.ReadLine()

        ' on demande l'âge
        Dim age As Integer
```

```

Dim ageOK As Boolean = False
Do While Not ageOK
    ' question
    Console.Out.WriteLine("Âge : ")
    ' lecture-vérification réponse
    Try
        age = Int32.Parse(System.Console.ReadLine())
        If age < 0 Then Throw New Exception
        ageOK = True
    Catch
        Console.Error.WriteLine("Age incorrect, recommencez...")
    End Try
Loop

' affichage final
Console.Out.WriteLine("Vous vous appelez [" & nom & "] et vous avez [" & age & "] ans")
End Sub
End Module

```

Quelques résultats d'exécution :

```

dos>console1
Nom : dupont
Âge : 23
Vous vous appelez dupont et vous avez 23 ans

```

```

dos>console1
Nom : dupont
Âge : xx
Age incorrect, recommencez...
Âge : 12
Vous vous appelez dupont et vous avez 12 ans

```

## 1.11 Passage de paramètres à une fonction

Nous nous intéressons ici au mode de passage des paramètres d'une fonction. Considérons la fonction :

```

Sub changeInt(ByVal a As Integer)
    a = 30
    Console.Out.WriteLine(("Paramètre formel a=" & a))
End Sub

```

Dans la définition de la fonction, *a* est appelé un paramètre formel. Il n'est là que pour les besoins de la définition de la fonction *changeInt*. Il aurait tout aussi bien pu s'appeler *b*. Considérons maintenant une utilisation de cette fonction :

```

Sub Main()
    Dim age As Integer = 20
    changeInt(age)
    Console.Out.WriteLine(("Paramètre effectif age=" & age))
End Sub

```

Ici dans l'instruction `changeInt(age)`, *age* est le paramètre effectif qui va transmettre sa valeur au paramètre formel *a*. Nous nous intéressons à la façon dont un paramètre formel récupère la valeur du paramètre effectif qui lui correspond.

### 1.11.1 Passage par valeur

L'exemple suivant nous montre que les paramètres d'une fonction/procédure sont par défaut passés par valeur : c'est à dire que la valeur du paramètre effectif est copiée dans le paramètre formel correspondant. On a deux entités distinctes. Si la fonction modifie le paramètre formel, le paramètre effectif n'est lui en rien modifié.

```

' options
Option Explicit On
Option Strict On

' passage de paramètres par valeur à une fonction
Imports System

Module param1
    Sub Main()
        Dim age As Integer = 20
        changeInt(age)
        Console.Out.WriteLine(("Paramètre effectif age=" & age))
    End Sub
End Module

```

```

End Sub

Sub changeInt(ByVal a As Integer)
    a = 30
    Console.Out.WriteLine("Paramètre formel a=" & a)
End Sub
End Module

```

Les résultats obtenus sont les suivants :

```

Paramètre formel a=30
Paramètre effectif age=20

```

La valeur 20 du paramètre effectif a été recopiée dans le paramètre formel *a*. Celui-ci a été ensuite modifié. Le paramètre effectif est lui resté inchangé. Ce mode de passage convient aux paramètres d'entrée d'une fonction.

## 1.11.2 Passage par référence

Dans un passage par référence, le paramètre effectif et le paramètre formel sont une seule et même entité. Si la fonction modifie le paramètre formel, le paramètre effectif est lui aussi modifié. En VB.NET, le paramètre formel doit être précédé du mot clé **ByRef**. Voici un exemple :

```

' options
Option Explicit On
Option Strict On

' passage de paramètres par valeur à une fonction
Imports System

Module param2
    Sub Main()
        Dim age As Integer = 20
        changeInt(age)
        Console.Out.WriteLine("Paramètre effectif age=" & age)
    End Sub

    Sub changeInt(ByRef a As Integer)
        a = 30
        Console.Out.WriteLine("Paramètre formel a=" & a)
    End Sub
End Module

```

et les résultats d'exécution :

```

Paramètre formel a=30
Paramètre effectif age=30

```

Le paramètre effectif a suivi la modification du paramètre formel. Ce mode de passage convient aux paramètres de sortie d'une fonction.

## 2. Classes, structures, interfaces

### 2.1 L' objet par l'exemple

#### 2.1.1 Généralités

Nous abordons maintenant, par l'exemple, la programmation objet. Un objet est une entité qui contient des données qui définissent son état (on les appelle des propriétés) et des fonctions (on les appelle des méthodes). Un objet est créé selon un modèle qu'on appelle une classe :

```
Public Class c1
  ' attributs
  Private p1 As type1
  Private p2 As type2
  ....

  ' méthode
  Public Sub m1(....)
  ...
End Sub

  ' méthode
  Public Function m2(...)
  ....
End Function
End Class
```

A partir de la classe *C1* précédente, on peut créer de nombreux objets *O1*, *O2*,... Tous auront les propriétés *p1*, *p2*,... et les méthodes *m3*, *m4*, ... Mais ils auront des valeurs différentes pour leurs propriétés *pi* ayant ainsi chacun un état qui leur est propre. Par analogie la déclaration

```
dim i, j as integer
```

créé deux objets (le terme est incorrect ici) de type (classe) *Integer*. Leur seule propriété est leur valeur. Si *O1* est un objet de type *C1*, *O1.p1* désigne la propriété *p1* de *O1* et *O1.m1* la méthode *m1* de *O1*. Considérons un premier modèle d'objet : la classe *personne*.

#### 2.1.2 Définition de la classe personne

La définition de la classe *personne* sera la suivante :

```
Public Class personne
  ' attributs
  Private prenom As String
  Private nom As String
  Private age As Integer

  ' méthode
  Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
    Me.prenom = P
    Me.nom = N
    Me.age = age
  End Sub

  ' méthode
  Public Sub identifie()
    Console.Out.WriteLine((prenom & "," & nom & "," & age))
  End Sub
End Class
```

Nous avons ici la définition d'une classe, donc d'un type de données. Lorsqu'on va créer des variables de ce type, on les appellera des objets ou des instances de classes. Une classe est donc un moule à partir duquel sont construits des objets. Les membres ou champs d'une classe peuvent être des données (attributs), des méthodes (fonctions), des propriétés. Les propriétés sont des méthodes particulières servant à connaître ou fixer la valeur d'attributs de l'objet. Ces champs peuvent être accompagnés de l'un des trois mots clés suivants :

<code>privé</code>	Un champ privé (private) n'est accessible que par les seules méthodes internes de la classe
<code>public</code>	Un champ public (public) est accessible par toute fonction définie ou non au sein de la classe

**protégé** | Un champ protégé (`protected`) n'est accessible que par les seules méthodes internes de la classe ou d'un objet dérivé (voir ultérieurement le concept d'héritage).

En général, les données d'une classe sont déclarées privées alors que ses méthodes et propriétés sont déclarées publiques. Cela signifie que l'utilisateur d'un objet (le programmeur) :

- n'aura pas accès directement aux données privées de l'objet
- pourra faire appel aux méthodes publiques de l'objet et notamment à celles qui donneront accès à ses données privées.

La syntaxe de déclaration d'une classe est la suivante :

```
public class classe
  private donnée ou méthode ou propriété privée
  public donnée ou méthode ou propriété publique
  protected donnée ou méthode ou propriété protégée
end class
```

L'ordre de déclaration des attributs *private*, *protected* et *public* est quelconque.

### 2.1.3 La méthode initialise

Revenons à notre classe [personne] déclarée comme :

```
Public Class personne
  ' attributs
  Private prenom As String
  Private nom As String
  Private age As Integer

  ' méthode
  Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
    Me.prenom = P
    Me.nom = N
    Me.age = age
  End Sub

  ' méthode
  Public Sub identifie()
    Console.Out.WriteLine((prenom & "," & nom & "," & age))
  End Sub
End Class
```

Quel est le rôle de la méthode *initialise* ? Parce que *nom*, *prenom* et *age* sont des données privées de la classe *personne*, les instructions :

```
dim p1 as personne p1
p1.prenom="Jean"
p1.nom="Dupont"
p1.age=30
```

sont illégales. Il nous faut initialiser un objet de type *personne* via une méthode publique. C'est le rôle de la méthode *initialise*. On écrira :

```
dim p1 as personne
p1.initialise("Jean", "Dupont", 30)
```

L'écriture *p1.initialise* est légale car *initialise* est d'accès public.

### 2.1.4 L'opérateur new

La séquence d'instructions

```
dim p1 as personne
p1.initialise("Jean", "Dupont", 30)
```

est incorrecte. L'instruction

```
dim p1 as personne
```

déclare *p1* comme une référence à un objet de type *personne*. Cet objet n'existe pas encore et donc *p1* n'est pas initialisé. C'est comme si on écrivait :

```
dim p1 as personne=nothing
```

où on indique explicitement avec le mot clé *nothing* que la variable *p1* ne référence encore aucun objet. Lorsqu'on écrit ensuite

```
p1.initialise("Jean","Dupont",30)
```

on fait appel à la méthode *initialise* de l'objet référencé par *p1*. Or cet objet n'existe pas encore et le compilateur signalera l'erreur. Pour que *p1* référence un objet, il faut écrire :

```
dim p1 as personne=new personne()
```

Cela a pour effet de créer un objet de type *personne* non encore initialisé : les attributs *nom* et *prenom* qui sont des références d'objets de type *String* auront la valeur *nothing*, et *age* la valeur *0*. Il y a donc une initialisation par défaut. Maintenant que *p1* référence un objet, l'instruction d'initialisation de cet objet

```
p1.initialise("Jean","Dupont",30)
```

est valide.

## 2.1.5 Le mot clé Me

Regardons le code de la méthode *initialise* :

```
Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
    Me.prenom = P
    Me.nom = N
    Me.age = age
End Sub
```

L'instruction *Me.prenom=P* signifie que l'attribut *prenom* de l'objet courant (*Me*) reçoit la valeur *P*. Le mot clé *Me* désigne l'objet courant : celui dans lequel se trouve la méthode exécutée. Comment le connaît-on ? Regardons comment se fait l'initialisation de l'objet référencé par *p1* dans le programme appelant :

```
p1.initialise("Jean","Dupont",30)
```

C'est la méthode *initialise* de l'objet *p1* qui est appelée. Lorsque dans cette méthode, on référence l'objet *Me*, on référence en fait l'objet *p1*. La méthode *initialise* aurait aussi pu être écrite comme suit :

```
Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
    prenom = P
    nom = N
    Me.age = age
End Sub
```

Lorsqu'une méthode d'un objet référence un attribut *A* de cet objet, l'écriture *Me.A* est implicite. On doit l'utiliser explicitement lorsqu'il y a conflit d'identificateurs. C'est le cas de l'instruction :

```
Me.age=age;
```

où *age* désigne un attribut de l'objet courant ainsi que le paramètre *age* reçu par la méthode. Il faut alors lever l'ambiguïté en désignant l'attribut *age* par *Me.age*.

## 2.1.6 Un programme de test

Voici un court programme de test :

```
Public Class personne
    ' attributs
    Private prenom As String
    Private nom As String
    Private age As Integer

    ' méthode
    Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
        Me.prenom = P
        Me.nom = N
        Me.age = age
    End Sub

    ' méthode
    Public Sub identifie()
        Console.WriteLine((prenom & "," & nom & "," & age))
    End Sub
End Class
```

```
End Sub
End Class
```

et les résultats obtenus :

```
dos>vbc personnel.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573
```

```
dos>personnel
Jean, Dupont, 30
```

## 2.1.7 Utiliser un fichier de classes compilées (assembly)

On notera que dans l'exemple précédent il y a deux classes dans notre programme de test : les classes *personne* et *test1*. Il y a une autre façon de procéder :

- on compile la classe **personne** dans un fichier particulier appelé un assemblage (assembly). Ce fichier a une extension *.dll*
- on compile la classe **test1** en référant l'assemblage qui contient la classe *personne*.

Les deux fichiers source deviennent les suivants :

```
test.vb      Module test1
              Sub Main()
                Dim p1 As New personne
                p1.initialise("Jean", "Dupont", 30)
                p1.identifie()
              End Sub
            End module

personne2.vb ' options
              Option Explicit On
              Option Strict On

              ' espaces de noms
              Imports System

              Public Class personne
                ' attributs
                Private prenom As String
                Private nom As String
                Private age As Integer

                ' méthode
                Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
                  Me.prenom = P
                  Me.nom = N
                  Me.age = age
                End Sub 'initialise

                ' méthode
                Public Sub identifie()
                  Console.Out.WriteLine((prenom & ", " & nom & ", " & age))
                End Sub 'identifie
              End Class 'personne
```

La classe *personne* est compilée par l'instruction suivante :

```
dos>vbc /t:library personne2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573
```

```
dos>dir
24/02/2004 16:50          509 personne2.vb
24/02/2004 16:49          143 test.vb
24/02/2004 16:50          3 584 personne2.dll
```

La compilation a produit un fichier *personne2.dll*. C'est l'option de compilation */t:library* qui indique de produire un fichier "assembly". Maintenant compilons le fichier *test.vb* :

```
dos>vbc /r:personne2.dll test.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573
```

```
dos>dir
24/02/2004 16:50          509 personne2.vb
24/02/2004 16:49          143 test.vb
24/02/2004 16:50          3 584 personne2.dll
24/02/2004 16:51          3 072 test.exe
```

L'option de compilation `/r:personne2.dll` indique au compilateur qu'il trouvera certaines classes dans le fichier `personne2.dll`. Lorsque dans le fichier source `test.vb`, il trouvera une référence à la classe `personne` classe non déclarée dans le source `test.vb`, il cherchera la classe `personne` dans les fichiers .dll référencés par l'option `/r`. Il trouvera ici la classe `personne` dans l'assemblage `personne2.dll`. On aurait pu mettre dans cet assemblage d'autres classes. Pour utiliser lors de la compilation plusieurs fichiers de classes compilées, on écrira :

```
vbc /r:fic1.dll /r:fic2.dll ... fichierSource.vb
```

L'exécution du programme `test1.exe` donne les résultats suivants :

```
dos>test
Jean, Dupont, 30
```

## 2.1.8 Une autre méthode initialise

Considérons toujours la classe `personne` et rajoutons-lui la méthode suivante :

```
' méthode
Public Sub initialise(ByVal P As personne)
    prenom = P.prenom
    nom = P.nom
    Me.age = P.age
End Sub
```

On a maintenant deux méthodes portant le nom `initialise` : c'est légal tant qu'elles admettent des paramètres différents. C'est le cas ici. Le paramètre est maintenant une référence `P` à une personne. Les attributs de la personne `P` sont alors affectés à l'objet courant (`Me`). On remarquera que la méthode `initialise` a un accès direct aux attributs de l'objet `P` bien que ceux-ci soient de type `private`. C'est toujours vrai : un objet `O1` d'une classe `C` a toujours accès aux attributs des objets de la même classe `C`. Voici un test de la nouvelle classe `personne`, celle-ci ayant été compilée dans `personne.dll` comme il a été expliqué précédemment :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

Module test1
    Sub Main()
        Dim p1 As New personne
        p1.initialise("Jean", "Dupont", 30)
        Console.Out.WriteLine("p1=")
        p1.identifie()
        Dim p2 As New personne
        p2.initialise(p1)
        Console.Out.WriteLine("p2=")
        p2.identifie()
    End Sub
End Module
```

et ses résultats :

```
p1=Jean, Dupont, 30
p2=Jean, Dupont, 30
```

## 2.1.9 Constructeurs de la classe personne

Un constructeur est une procédure qui porte le nom **New** et qui est appelée lors de la création de l'objet. On s'en sert généralement pour l'initialiser. Si une classe a un constructeur acceptant `n` arguments `argi`, la déclaration et l'initialisation d'un objet de cette classe pourra se faire sous la forme :

```
dim objet as classe =new classe(arg1,arg2, ... argn)
ou
dim objet as classe
...
objet=new classe(arg1,arg2, ... argn)
```

Lorsqu'une classe a un ou plusieurs constructeurs, l'un de ces constructeurs doit être obligatoirement utilisé pour créer un objet de cette classe. Si une classe *C* n'a aucun constructeur, elle en a un par défaut qui est le constructeur sans paramètres : *public New()*. Les attributs de l'objet sont alors initialisés avec des valeurs par défaut. C'est ce qui s'est passé dans les programmes précédents, où on avait écrit :

```
dim p1 as personne
p1=new personne
```

Créons deux constructeurs à notre classe *personne* :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' la classe personne
Public Class personne
    ' attributs
    Private prenom As String
    Private nom As String
    Private age As Integer

    ' constructeurs
    Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
        initialise(P, N, age)
    End Sub

    Public Sub New(ByVal P As personne)
        initialise(P)
    End Sub

    ' méthodes d'initialisation de l'objet
    Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
        Me.prenom = P
        Me.nom = N
        Me.age = age
    End Sub

    Public Sub initialise(ByVal P As personne)
        prenom = P.prenom
        nom = P.nom
        Me.age = P.age
    End Sub

    ' méthode
    Public Sub identifie()
        Console.Out.WriteLine((prenom & "," & nom & "," & age))
    End Sub
End Class
```

Nos deux constructeurs se contentent de faire appel aux méthodes *initialise* correspondantes. On rappelle que lorsque dans un constructeur, on trouve la notation *initialise(P)* par exemple, le compilateur traduit par *Me.initialise(P)*. Dans le constructeur, la méthode *initialise* est donc appelée pour travailler sur l'objet référencé par *Me*, c'est à dire l'objet courant, celui qui est en cours de construction. Voici un court programme de test :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
    Sub Main()
        Dim p1 As New personne("Jean", "Dupont", 30)
        Console.Out.Write("p1=")
        p1.identifie()
        Dim p2 As New personne(p1)
        Console.Out.Write("p2=")
        p2.identifie()
    End Sub
End Module
```

et les résultats obtenus :

```
p1=Jean, Dupont, 30
p2=Jean, Dupont, 30
```

## 2.1.10 Les références d'objets

Nous utilisons toujours la même classe *personne*. Le programme de test devient le suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    ' p1
    Dim p1 As New personne("Jean", "Dupont", 30)
    Console.Out.Write("p1=")
    p1.identifie()

    ' p2 référence le même objet que p1
    Dim p2 As personne = p1
    Console.Out.Write("p2=")
    p2.identifie()

    ' p3 référence un objet qui sera une copie de l'objet référencé par p1
    Dim p3 As New personne(p1)
    Console.Out.Write("p3=")
    p3.identifie()

    ' on change l'état de l'objet référencé par p1
    p1.initialise("Micheline", "Benoît", 67)
    Console.Out.Write("p1=")
    p1.identifie()

    ' comme p2=p1, l'objet référencé par p2 a du changer d'état
    Console.Out.Write("p2=")
    p2.identifie()

    ' comme p3 ne référence pas le même objet que p1, l'objet référencé par p3 n'a pas du changer
    Console.Out.Write("p3=")
    p3.identifie()
End Sub
End Module
```

Les résultats obtenus sont les suivants :

```
p1=Jean, Dupont, 30
p2=Jean, Dupont, 30
p3=Jean, Dupont, 30
p1=Micheline, Benoît, 67
p2=Micheline, Benoît, 67
p3=Jean, Dupont, 30
```

Lorsqu'on déclare la variable *p1* par

```
dim p1 as personne=new personne("Jean", "Dupont", 30)
```

*p1* référence l'objet *personne("Jean","Dupont",30)* mais n'est pas l'objet lui-même. En C, on dirait que c'est un pointeur, c.a.d. l'adresse de l'objet créé. Si on écrit ensuite :

```
p1=nothing
```

Ce n'est pas l'objet *personne("Jean","Dupont",30)* qui est modifié, c'est la référence *p1* qui change de valeur. L'objet *personne("Jean","Dupont",30)* sera "perdu" s'il n'est référencé par aucune autre variable.

Lorsqu'on écrit :

```
dim p2 as personne=p1
```

on initialise le pointeur *p2* : il "pointe" sur le même objet (il désigne le même objet) que le pointeur *p1*. Ainsi si on modifie l'objet "pointé" (ou référencé) par *p1*, on modifie celui référencé par *p2*.

Lorsqu'on écrit :

```
dim p3 as personne =new personne(p1);
```

il y a création d'un nouvel objet, copie de l'objet référencé par *p1*. Ce nouvel objet sera référencé par *p3*. Si on modifie l'objet "pointé" (ou référencé) par *p1*, on ne modifie en rien celui référencé par *p3*. C'est ce que montrent les résultats obtenus.

## 2.1.11 Les objets temporaires

Dans une expression, on peut faire appel explicitement au constructeur d'un objet : celui-ci est construit, mais nous n'y avons pas accès (pour le modifier par exemple). Cet objet temporaire est construit pour les besoins d'évaluation de l'expression puis abandonné. L'espace mémoire qu'il occupait sera automatiquement récupéré ultérieurement par un programme appelé "ramasse-miettes" dont le rôle est de récupérer l'espace mémoire occupé par des objets qui ne sont plus référencés par des données du programme. Considérons le nouveau programme de test suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    Dim p As New personne(New personne("Jean", "Dupont", 30))
    p.identifie()
End Sub
End Module
```

et modifions les constructeurs de la classe *personne* afin qu'ils affichent un message :

```
' constructeurs
Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
    Console.Out.WriteLine("Constructeur personne(String, String, integer)")
    initialise(P, N, age)
End Sub

Public Sub New(ByVal P As personne)
    Console.Out.WriteLine("Constructeur personne(personne)")
    initialise(P)
End Sub
```

Nous obtenons les résultats suivants :

```
dos>test
Constructeur personne(String, String, integer)
Constructeur personne(personne)
Jean,Dupont,30
```

montrant la construction successive des deux objets temporaires.

## 2.1.12 Méthodes de lecture et d'écriture des attributs privés

Nous rajoutons à la classe *personne* les méthodes nécessaires pour lire ou modifier l'état des attributs des objets :

```
Imports System

Public Class personne

    ' attributs
    Private prenom As [String]
    Private nom As [String]
    Private age As Integer

    ' constructeurs
    Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
        Me.prenom = P
        Me.nom = N
        Me.age = age
    End Sub

    Public Sub New(ByVal P As personne)
        Me.prenom = P.prenom
```

```

    Me.nom = P.nom
    Me.age = P.age
End Sub

' identifie
Public Sub identifie()
    Console.Out.WriteLine((prenom + "," + nom + "," + age))
End Sub

' accesseurs
Public Function getPrenom() As [String]
    Return prenom
End Function

Public Function getNom() As [String]
    Return nom
End Function

Public Function getAge() As Integer
    Return age
End Function

' modifieurs
Public Sub setPrenom(ByVal P As [String])
    Me.prenom = P
End Sub

Public Sub setNom(ByVal N As [String])
    Me.nom = N
End Sub

Public Sub setAge(ByVal age As Integer)
    Me.age = age
End Sub
End Class

```

Nous testons la nouvelle classe avec le programme suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' pg de test
Public Module test
    Sub Main()
        Dim P As New personne("Jean", "Michelin", 34)
        Console.Out.WriteLine(("P=(" & P.getPrenom() & "," & P.getNom() & "," & P.getAge() & ")"))
        P.setAge(56)
        Console.Out.WriteLine(("P=(" & P.getPrenom() & "," & P.getNom() & "," & P.getAge() & ")"))
    End Sub
End Module

```

et nous obtenons les résultats suivants :

```

P=(Jean,Michelin,34)
P=(Jean,Michelin,56)

```

## 2.1.13 Les propriétés

Il existe une autre façon d'avoir accès aux attributs d'une classe c'est de créer des propriétés. Celles-ci nous permettent de manipuler des attributs privés comme s'ils étaient publics. Considérons la classe *personne* suivante où les accesseurs et modifieurs précédents ont été remplacés par des **propriétés** en lecture et écriture :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' classe personne
Public Class personne

' attributs
    Private _prenom As [String]

```

```

Private _nom As [String]
Private _age As Integer

' constructeurs
Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
    Me._prenom = P
    Me._nom = N
    Me._age = age
End Sub

Public Sub New(ByVal P As personne)
    Me._prenom = P._prenom
    Me._nom = P._nom
    Me._age = P._age
End Sub

' identifie
Public Sub identifie()
    Console.WriteLine((_prenom & "," & _nom & "," & _age))
End Sub

' propriétés
Public Property prenom() As String
    Get
        Return _prenom
    End Get
    Set(ByVal Value As String)
        _prenom = Value
    End Set
End Property

Public Property nom() As String
    Get
        Return _nom
    End Get
    Set(ByVal Value As String)
        _nom = Value
    End Set
End Property

Public Property age() As Integer
    Get
        Return _age
    End Get
    Set(ByVal Value As Integer)
        ' age valide ?
        If Value >= 0 Then
            _age = Value
        Else
            Throw New Exception("âge (" & Value & ") invalide")
        End If
    End Set
End Property
End Class

```

Une propriété `Property` permet de lire (**get**) ou de fixer (**set**) la valeur d'un attribut. Dans notre exemple, nous avons préfixé les noms des attributs du signe `_` afin que les propriétés portent le nom des attributs primitifs. En effet, une propriété ne peut porter le même nom que l'attribut qu'elle gère car alors il y a un conflit de noms dans la classe. Nous avons donc appelé nos attributs `_prenom`, `_nom`, `_age` et modifié les constructeurs et méthodes en conséquence. Nous avons ensuite créé trois propriétés `nom`, `prenom` et `age`. Une propriété est déclarée comme suit :

```

Public Property nom() As Type
    Get
    ...
    End Get
    Set(ByVal Value As Type)
    ...
    End Set
End Property

```

où `Type` doit être le type de l'attribut géré par la propriété. Elle peut avoir deux méthodes appelées **get** et **set**. La méthode **get** est habituellement chargée de rendre la valeur de l'attribut qu'elle gère (elle pourrait rendre autre chose, rien ne l'empêche). La méthode **set** reçoit un paramètre appelé **value** qu'elle affecte normalement à l'attribut qu'elle gère. Elle peut en profiter pour faire des vérifications sur la validité de la valeur reçue et éventuellement lancer une exception si la valeur se révèle invalide. C'est ce qui est fait ici pour l'âge.

Comment ces méthodes **get** et **set** sont-elles appelées ? Considérons le programme de test suivant :

```
' options
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    Dim P As New personne("Jean", "Michelin", 34)
    Console.Out.WriteLine("P=(" & P.prenom & "," & P.nom & "," & P.age & ")")
    P.age = 56
    Console.Out.WriteLine("P=(" & P.prenom & "," & P.nom & "," & P.age & ")")
    Try
        P.age = -4
    Catch ex As Exception
        Console.Error.WriteLine(ex.Message)
    End Try
End Sub
End Module
```

Dans l'instruction

```
Console.Out.WriteLine("P=(" & P.prenom & "," & P.nom & "," & P.age & ")")
```

on cherche à avoir les valeurs des propriétés *prenom*, *nom* et *age* de la personne P. C'est la méthode **get** de ces propriétés qui est alors appelée et qui rend la valeur de l'attribut qu'elles gèrent.

Dans l'instruction

```
P.age = 56
```

on veut fixer la valeur de la propriété *age*. C'est alors la méthode set de cette propriété qui est alors appelée. Elle recevra 56 dans son paramètre **value**.

Une propriété **P** d'une classe **C** qui ne définirait que la méthode **get** est dite en lecture seule. Si c est un objet de classe C, l'opération *c.P= valeur* sera alors refusée par le compilateur.

L'exécution du programme de test précédent donne les résultats suivants :

```
P=(Jean,Michelin,34)
P=(Jean,Michelin,56)
âge (-4) invalide
```

Les propriétés nous permettent donc de manipuler des attributs privés comme s'ils étaient publics.

## 2.1.14 Les méthodes et attributs de classe

Supposons qu'on veuille compter le nombre d'objets [personne] créées dans une application. On peut soi-même gérer un compteur mais on risque d'oublier les objets temporaires qui sont créés ici ou là. Il semblerait plus sûr d'inclure dans les constructeurs de la classe [personne], une instruction incrémentant un compteur. Le problème est de passer une référence de ce compteur afin que le constructeur puisse l'incrémenter : il faut leur passer un nouveau paramètre. On peut aussi inclure le compteur dans la définition de la classe. Comme c'est un attribut de la classe elle-même et non d'un objet particulier de cette classe, on le déclare différemment avec le mot clé *Shared* :

```
Private Shared _nbPersonnes As Long = 0
```

Pour le référencer, on écrit *personne.\_nbPersonnes* pour montrer que c'est un attribut de la classe *personne* elle-même. Ici, nous avons créé un attribut privé auquel on n'aura pas accès directement en-dehors de la classe. On crée donc une propriété publique pour donner accès à l'attribut de classe *nbPersonnes*. Pour rendre la valeur de *nbPersonnes* la méthode *get* de cette propriété n'a pas besoin d'un objet *personne* particulier : en effet *\_nbPersonnes* n'est pas l'attribut d'un objet particulier, il est l'attribut de toute une classe. Aussi a-t-on besoin d'une propriété déclarée elle-aussi *Shared* :

```
Public Shared ReadOnly Property nbPersonnes() As Long
    Get
        Return _nbPersonnes
    End Get
End Property
```

qui de l'extérieur sera appelée avec la syntaxe *personne.nbPersonnes*. La propriété est déclarée en lecture seule (ReadOnly) car elle ne propose pas de méthode set. Voici un exemple. La classe *personne* devient la suivante :

```

Option Explicit On
Option Strict On

' espaces de noms
Imports System

' classe
Public Class personne
' attributs de classe
Private Shared _nbPersonnes As Long = 0

' attributs d'instance
Private _prenom As [String]
Private _nom As [String]
Private _age As Integer

' constructeurs
Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
' une personne de plus
_nbPersonnes += 1
Me._prenom = P
Me._nom = N
Me._age = age
End Sub

Public Sub New(ByVal P As personne)
' une personne de plus
_nbPersonnes += 1
Me._prenom = P._prenom
Me._nom = P._nom
Me._age = P._age
End Sub

' identifie
Public Sub identifie()
Console.Out.WriteLine((_prenom & "," & _nom & "," & _age))
End Sub

' propriété de classe
Public Shared ReadOnly Property nbPersonnes() As Long
Get
Return _nbPersonnes
End Get
End Property

' propriétés d'instance
Public Property prenom() As String
Get
Return _prenom
End Get
Set(ByVal Value As String)
_prenom = Value
End Set
End Property

Public Property nom() As String
Get
Return _nom
End Get
Set(ByVal Value As String)
_nom = Value
End Set
End Property

Public Property age() As Integer
Get
Return _age
End Get
Set(ByVal Value As Integer)
' age valide ?
If Value >= 0 Then
_age = Value
Else
Throw New Exception("âge (" & Value & ") invalide")
End If
End Set
End Property
End Class

```

Avec le programme suivant :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    Dim p1 As New personne("Jean", "Dupont", 30)
    Dim p2 As New personne(p1)
    Console.Out.WriteLine("Nombre de personnes créées : " & personne.nbPersonnes)
End Sub
End Module

```

on obtient les résultats suivants :

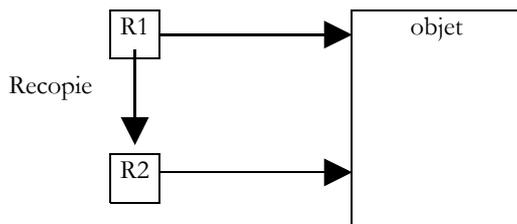
```

Nombre de personnes créées : 2

```

## 2.1.15 Passage d'un objet à une fonction

Nous avons déjà dit que par défaut VB.NET passait les paramètres effectifs d'une fonction par valeur : les valeurs des paramètres effectifs sont recopiées dans les paramètres formels. Dans le cas d'un objet, il ne faut pas se laisser tromper par l'abus de langage qui est fait systématiquement en parlant d'objet au lieu de référence d'objet. Un objet n'est manipulé que via une référence (un pointeur) sur lui. Ce qui est donc transmis à une fonction, n'est pas l'objet lui-même mais une référence sur cet objet. C'est donc la valeur de la référence et non la valeur de l'objet lui-même qui est dupliquée dans le paramètre formel : il n'y a pas construction d'un nouvel objet. Si une référence d'objet R1 est transmise à une fonction, elle sera recopiée dans le paramètre formel correspondant R2. Aussi les références R2 et R1 désignent-elles le même objet. Si la fonction modifie l'objet pointé par R2, elle modifie évidemment celui référencé par R1 puisque c'est le même.



C'est ce que montre l'exemple suivant :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    ' une personne p1
    Dim p1 As New personne("Jean", "Dupont", 30)

    ' affichage p1
    Console.Out.Write("Paramètre effectif avant modification : ")
    p1.identifie()

    ' modification p1
    modifie(p1)

    ' affichage p1
    Console.Out.Write("Paramètre effectif après modification : ")
    p1.identifie()
End Sub

Sub modifie(ByVal P As personne)
    ' affichage personne P
    Console.Out.Write("Paramètre formel avant modification : ")
    P.identifie()
    ' modification P
    P.prenom = "Sylvie"
    P.nom = "Vartan"
    P.age = 52

```

```

' affichage P
Console.Out.WriteLine("Paramètre formel après modification : ")
P.identifie()
End Sub
End Module

```

Les résultats obtenus sont les suivants :

```

Paramètre effectif avant modification : Jean,Dupont,30
Paramètre formel avant modification : Jean,Dupont,30
Paramètre formel après modification : Sylvie,Vartan,52
Paramètre effectif après modification : Sylvie,Vartan,52

```

On voit qu'il n'y a construction que d'un objet : celui de la personne *p1* de la procédure *Main* et que l'objet a bien été modifié par la fonction *modifie*.

## 2.1.16 Un tableau de personnes

Un objet est une donnée comme une autre et à ce titre plusieurs objets peuvent être rassemblés dans un tableau :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    ' un tableau de personnes
    Dim amis(2) As personne
    amis(0) = New personne("Jean", "Dupont", 30)
    amis(1) = New personne("Sylvie", "Vartan", 52)
    amis(2) = New personne("Neil", "Armstrong", 66)
    ' affichage
    Console.Out.WriteLine("-----")
    Dim i As Integer
    For i = 0 To amis.Length - 1
        amis(i).identifie()
    Next i
End Sub
End Module

```

L'instruction `Dim amis(2) As personne` crée un tableau de 3 éléments de type *personne*. Ces 3 éléments sont initialisés ici avec la valeur *nothing*, c.a.d. qu'ils ne référencent aucun objet. De nouveau, par abus de langage, on parle de tableau d'objets alors que ce n'est qu'un tableau de références d'objets. La création du tableau d'objets, qui est un objet lui-même ne crée aucun objet du type de ses éléments : il faut le faire ensuite. On obtient les résultats suivants :

```

Jean,Dupont,30
Sylvie,Vartan,52
Neil,Armstrong,66

```

## 2.2 L'héritage par l'exemple

### 2.2.1 Généralités

Nous abordons ici la notion d'héritage. Le but de l'héritage est de "personnaliser" une classe existante pour qu'elle satisfasse à nos besoins. Supposons qu'on veuille créer une classe *enseignant* : un enseignant est une personne particulière. Il a des attributs qu'une autre personne n'aura pas : la matière qu'il enseigne par exemple. Mais il a aussi les attributs de toute personne : prénom, nom et âge. Un enseignant fait donc pleinement partie de la classe *personne* mais a des attributs supplémentaires. Plutôt que d'écrire une classe *enseignant* à partir de rien, on préférerait reprendre l'acquis de la classe *personne* qu'on adapterait au caractère particulier des enseignants. C'est le concept d'**héritage** qui nous permet cela. Pour exprimer que la classe *enseignant* hérite des propriétés de la classe *personne*, on écrira :

```

Public Class enseignant
    Inherits personne

```

On notera la syntaxe particulière sur deux lignes. La classe *personne* est appelée la classe parent (ou mère) et la classe *enseignant* la classe dérivée (ou fille). Un objet *enseignant* a toutes les qualités d'un objet *personne* : il a les mêmes attributs et les mêmes méthodes.

Ces attributs et méthodes de la classe parent ne sont pas répétées dans la définition de la classe fille : on se contente d'indiquer les attributs et méthodes rajoutés par la classe fille. Nous supposons que la classe *personne* est définie comme suit :

```
' options
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' classe personne
Public Class personne

' attributs de classe
Private Shared _nbPersonnes As Long = 0

' attributs d'instance
Private _prenom As [String]
Private _nom As [String]
Private _age As Integer

' constructeurs
Public Sub New(ByVal P As [String], ByVal N As [String], ByVal age As Integer)
' une personne de plus
_nbPersonnes += 1
' construction
Me._prenom = P
Me._nom = N
Me._age = age
' suivi
Console.Out.WriteLine("Construction personne(string, string, int)")
End Sub

Public Sub New(ByVal P As personne)
' une personne de plus
_nbPersonnes += 1
' construction
Me._prenom = P._prenom
Me._nom = P._nom
Me._age = P._age
' suivi
Console.Out.WriteLine("Construction personne(string, string, int)")
End Sub

' propriété de classe
Public Shared ReadOnly Property nbPersonnes() As Long
Get
Return _nbPersonnes
End Get
End Property

' propriétés d'instance
Public Property prenom() As String
Get
Return _prenom
End Get
Set(ByVal Value As String)
_prenom = Value
End Set
End Property

Public Property nom() As String
Get
Return _nom
End Get
Set(ByVal Value As String)
_nom = Value
End Set
End Property

Public Property age() As Integer
Get
Return _age
End Get
Set(ByVal Value As Integer)
' age valide ?
If Value >= 0 Then
_age = Value
```

```

Else
    Throw New Exception("âge (" & Value & ") invalide")
End If
End Set
End Property

Public ReadOnly Property identite() As String
Get
    Return "personne(" & _prenom & "," & _nom & "," & age & ")"
End Get
End Property
End Class

```

La méthode *identifie* a été remplacée par la propriété *identité* en lecture seule et qui identifie la personne. Nous créons une classe *enseignant* héritant de la classe *personne* :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Class enseignant
    Inherits personne
    ' attributs
    Private _section As Integer

    ' constructeur
    Public Sub New(ByVal P As String, ByVal N As String, ByVal age As Integer, ByVal section As Integer)
        MyBase.New(P, N, age)
        Me._section = section
    ' suivi
        Console.Out.WriteLine("Construction enseignant(string,string,int,int)")
    End Sub

    ' propriété section
    Public Property section() As Integer
    Get
        Return _section
    End Get
    Set(ByVal Value As Integer)
        _section = Value
    End Set
End Property
End Class

```

La classe *enseignant* rajoute aux méthodes et attributs de la classe *personne* :

- un attribut *section* qui est le n° de section auquel appartient l'enseignant dans le corps des enseignants (une section par discipline en gros)
- un nouveau constructeur permettant d'initialiser tous les attributs d'un enseignant

La déclaration

```
Public Class enseignant
    Inherits personne
```

indique que la classe *enseignant* dérive de la classe *personne*.

## 2.2.2 Construction d'un objet enseignant

Le constructeur de la classe *enseignant* est le suivant :

```

' constructeur
Public Sub New(ByVal P As String, ByVal N As String, ByVal age As Integer, ByVal section As Integer)
    MyBase.New(P, N, age)
    Me._section = section
    ' suivi
    Console.Out.WriteLine("Construction enseignant(string,string,int,int)")
End Sub

```

La déclaration

```
Public Sub New(ByVal P As String, ByVal N As String, ByVal age As Integer, ByVal section As Integer)
```

déclare que le constructeur reçoit quatre paramètres *P*, *N*, *age*, *section*. Elle doit en passer trois (*P,N,age*) à sa classe de base, ici la classe *personne*. On sait que cette classe a un constructeur *personne(string, string, int)* qui va permettre de construire une personne avec les paramètres passés (*P,N,age*). La classe [enseignant] passe les paramètres (*P,N,age*) à sa classe de base de la façon suivante :

```
MyBase.New(P, N, age)
```

Une fois la construction de la classe de base terminée, la construction de l'objet *enseignant* se poursuit par l'exécution du corps du constructeur :

```
Me._section = section
```

En résumé, le constructeur d'une classe dérivée :

- passe à sa classe de base les paramètres dont elle a besoin pour se construire
- utilise les autres paramètres pour initialiser les attributs qui lui sont propres

On aurait pu préférer écrire :

```
Public Sub New(ByVal P As String, ByVal N As String, ByVal age As Integer, ByVal section As Integer)
    Me._prenom = P
    Me._nom = N
    Me._age = age
    Me._section = section
    ' suivi
    Console.Out.WriteLine("Construction enseignant(string,string,int,int)")
End Sub
```

C'est impossible. La classe *personne* a déclaré privés (*private*) ses trois champs *\_prenom*, *\_nom* et *\_age*. Seuls des objets de la même classe ont un accès direct à ces champs. Tous les autres objets, y compris des objets fils comme ici, doivent passer par des méthodes publiques pour y avoir accès. Cela aurait été différent si la classe *personne* avait déclaré protégés (*protected*) les trois champs : elle autorisait alors des classes dérivées à avoir un accès direct aux trois champs. Dans notre exemple, utiliser le constructeur de la classe parent était donc la bonne solution et c'est la méthode habituelle : lors de la construction d'un objet fils, on appelle d'abord le constructeur de l'objet parent puis on complète les initialisations propres cette fois à l'objet fils (*section* dans notre exemple).

Compilons les classes *personne* et *enseignant* dans des assemblages :

```
dos>vbc /t:library personne.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>vbc /r:personne.dll /t:library enseignant.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>dir
25/02/2004 10:08          1 828 personne.vb
25/02/2004 10:11          675 enseignant.vb
25/02/2004 10:12          223 test.vb
25/02/2004 10:16          4 096 personne.dll
25/02/2004 10:16          3 584 enseignant.dll
```

On remarquera que pour compiler la classe fille *enseignant*, il a fallu référencer le fichier *personne.dll* qui contient la classe *personne*. Tentons un premier programme de test :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
    Sub Main()
        Console.Out.WriteLine(New enseignant("Jean", "Dupont", 30, 27).identite)
    End Sub
End Module
```

Ce programme se contente de créer un objet *enseignant* (*new*) et de l'identifier. La classe *enseignant* n'a pas de méthode *identite* mais sa classe parent en a une qui de plus est publique : elle devient par héritage une méthode publique de la classe *enseignant*. Les résultats obtenus sont les suivants :

```
dos>vbc /r:personne.dll /r:enseignant.dll test.vb
```

```
dos>test
Construction personne(string, string, int)
Construction enseignant(string, string,int,int)
personne (Jean,Dupont,30)
```

On voit que :

- un objet *personne* a été construit avant l'objet *enseignant*
- l'identité obtenue est celle de l'objet *personne*

## 2.2.3 Surcharge d'une méthode ou d'une propriété

Dans l'exemple précédent, nous avons eu l'identité de la partie *personne* de l'enseignant mais il manque certaines informations propres à la classe *enseignant* (la section). On est donc amené à écrire une propriété permettant d'identifier l'enseignant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Class enseignant
    Inherits personne
    ' attributs
    Private _section As Integer

    ' constructeur
    Public Sub New(ByVal P As String, ByVal N As String, ByVal age As Integer, ByVal section As Integer)
        MyBase.New(P, N, age)
        Me._section = section
    ' suivi
        Console.Out.WriteLine("Construction enseignant(string,string,int,int)")
    End Sub

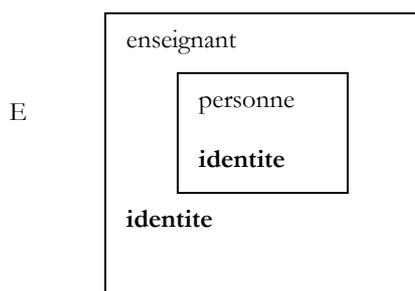
    ' propriété section
    Public Property section() As Integer
        Get
            Return _section
        End Get
        Set(ByVal Value As Integer)
            _section = Value
        End Set
    End Property

    ' surcharge propriété identité
    Public Shadows ReadOnly Property identite() As String
        Get
            Return "enseignant(" & MyBase.identite & "," & _section & ")"
        End Get
    End Property
End Class
```

La méthode *identite* de la classe *enseignant* s'appuie sur la méthode *identite* de sa classe mère (`MyBase.identite`) pour afficher sa partie "*personne*" puis complète avec le champ *\_section* qui est propre à la classe *enseignant*. Notons la déclaration de la propriété *identite* :

```
Public Shadows ReadOnly Property identite() As String
```

qui indique que la propriété *identite* "cache" la méthode de même nom qui pourrait exister dans la classe parent. Soit un objet *enseignant* E. Cet objet contient en son sein un objet *personne* :



La propriété identité est définie à la fois dans la classe *enseignant* et sa classe mère *personne*. Dans la classe fille *enseignant*, la propriété *identité* doit être précédée du mot clé **shadows** pour indiquer qu'on redéfinit une nouvelle propriété *identité* pour la classe *enseignant*.

```
Public Shadows ReadOnly Property identite() As String
```

La classe *enseignant* dispose maintenant de deux propriétés *identité* :

- celle héritée de la classe parent *personne*
- la sienne propre

Si *E* est un objet *enseignant*, *E.identite* désigne la méthode *identité* de la classe *enseignant*. On dit que la propriété *identité* de la classe mère est "redéfinie" par la propriété *identité* de la classe fille. De façon générale, si *O* est un objet et *M* une méthode, pour exécuter la méthode *O.M*, le système cherche une méthode *M* dans l'ordre suivant :

- dans la classe de l'objet *O*
- dans sa classe mère s'il en a une
- dans la classe mère de sa classe mère si elle existe
- etc...

L'héritage permet donc de redéfinir dans la classe fille des méthodes/propriétés de même nom dans la classe mère. C'est ce qui permet d'adapter la classe fille à ses propres besoins. Associée au polymorphisme que nous allons voir un peu plus loin, la surcharge de méthodes/propriétés est le principal intérêt de l'héritage. Considérons le même exemple que précédemment :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
Sub Main()
    Console.Out.WriteLine(New enseignant("Jean", "Dupont", 30, 27).identite)
End Sub
End Module
```

Les résultats obtenus sont cette fois les suivants :

```
Construction personne(string, string, int)
Construction enseignant(string, string, int, int)
enseignant(personne(Jean, Dupont, 30), 27)
```

## 2.2.4 Le polymorphisme

Considérons une lignée de classes :  $C_0 \rightarrow C_1 \rightarrow C_2 \rightarrow \dots \rightarrow C_n$  où  $C_i \rightarrow C_j$  indique que la classe  $C_j$  est dérivée de la classe  $C_i$ . Cela entraîne que la classe  $C_j$  a toutes les caractéristiques de la classe  $C_i$  plus d'autres. Soient des objets  $O_i$  de type  $C_i$ . Il est légal d'écrire :  $O_i = O_j$  avec  $j > i$

En effet, par héritage, la classe  $C_j$  a toutes les caractéristiques de la classe  $C_i$  plus d'autres. Donc un objet  $O_j$  de type  $C_j$  contient en lui un objet de type  $C_i$ . L'opération

$O_i = O_j$

fait que  $O_i$  est une référence à l'objet de type  $C_i$  contenu dans l'objet  $O_j$ .

Le fait qu'une variable  $O_i$  de classe  $C_i$  puisse en fait référencer non seulement un objet de la classe  $C_i$  mais en fait tout objet dérivé de la classe  $C_i$  est appelé **polymorphisme** : la faculté pour une variable de référencer différents types d'objets. Prenons un exemple et considérons la fonction suivante indépendante de toute classe :

```
Sub affiche(ByVal p As personne)
```

On pourra aussi bien écrire

```
dim p as personne
...
affiche(p);
```

que

```
dim e as enseignant
...
affiche(e);
```

Dans ce dernier cas, le paramètre formel de type *personne* de la fonction *affiche* va recevoir une valeur de type *enseignant*. Comme le type *enseignant* dérive du type *personne*, c'est légal.

## 2.2.5 Redéfinition et polymorphisme

Complétons notre procédure *affiche* :

```
Sub affiche(ByVal p As personne)
    ' affiche identité de p
    Console.Out.WriteLine(p.identite)
End Sub
```

La méthode *p.identite* rend une chaîne de caractères identifiant l'objet *personne*. Que se passe-t-il dans le cas de notre exemple précédent dans le cas d'un objet *enseignant* :

```
Dim e As New enseignant("Lucile", "Dumas", 56, 61)
affiche(e)
```

Regardons l'exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
    Sub Main()
        ' un enseignant
        Dim e As New enseignant("Lucile", "Dumas", 56, 61)
        affiche(e)
        ' une personne
        Dim p As New personne("Jean", "Dupont", 30)
        affiche(p)
    End Sub

    ' affiche
    Sub affiche(ByVal p As personne)
        ' affiche identité de p
        Console.Out.WriteLine(p.identite)
    End Sub
End Module
```

Les résultats obtenus sont les suivants :

```
Construction personne(string, string, int)
Construction enseignant(string,string,int,int)
personne(Lucile,Dumas,56)
Construction personne(string, string, int)
personne(Jean,Dupont,30)
```

L'exécution montre que l'instruction *p.identite* a exécuté à chaque fois la propriété *identite* d'une *personne*, la personne contenue dans l'*enseignant* *e*, puis la *personne* *p* elle-même. Elle ne s'est pas adaptée à l'objet réellement passé en paramètre à *affiche*. On aurait préféré avoir l'identité complète de l'*enseignant* *e*. Il aurait fallu pour cela que la notation *p.identite* référence la propriété *identite* de l'objet réellement pointé par *p* plutôt que la propriété *identite* de partie "*personne*" de l'objet réellement pointé par *p*. Il est possible d'obtenir ce résultat en déclarant *identite* comme une propriété **redéfinissable** (**overridable**) dans la classe de base *personne* :

```
Public Overridable ReadOnly Property identite() As String
    Get
        Return "personne(" & _prenom & ", " & _nom & ", " & age & ")"
    End Get
End Property
```

Le mot clé **overridable** fait de *identite* une propriété réfinissable ou virtuelle. Ce mot clé peut s'appliquer également aux méthodes. Les classes filles qui redéfinissent une propriété ou méthode virtuelle doivent utiliser alors le mot clé **overrides** au lieu de **shadow**s pour qualifier leur propriété/méthode redéfinie. Ainsi dans la classe *enseignant*, la propriété *identite* est définie comme suit :

```
' surcharge propriété identité
Public Overrides ReadOnly Property identite() As String
    Get
        Return "enseignant(" & MyBase.identite & ", " & _section & ")"
    End Get
End Property
```

Le programme de test :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
Sub Main()
    ' un enseignant
    Dim e As New enseignant("Lucile", "Dumas", 56, 61)
    affiche(e)
    ' une personne
    Dim p As New personne("Jean", "Dupont", 30)
    affiche(p)
End Sub

' affiche
Sub affiche(ByVal p As personne)
    ' affiche identité de p
    Console.Out.WriteLine(p.identite)
End Sub
End Module
```

produit alors les résultats suivants :

```
Construction personne(string, string, int)
Construction enseignant(string, string, int, int)
enseignant(personne(Lucile, Dumas, 56), 61)
Construction personne(string, string, int)
personne(Jean, Dupont, 30)
```

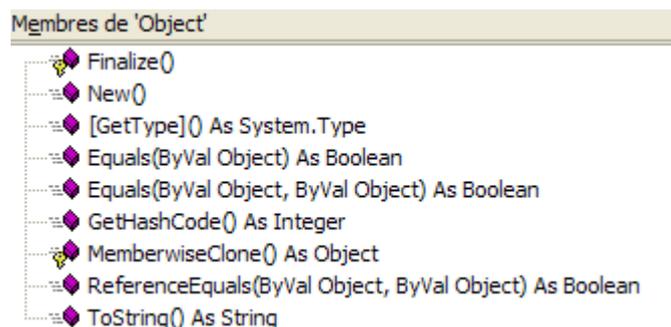
Cette fois-ci, on a bien eu l'identité complète de l'enseignant. Redéfinissons maintenant une méthode plutôt qu'une propriété. La classe *object* est la classe "mère" de toutes les classes VB.NET. Ainsi lorsqu'on écrit :

```
public class personne
```

on écrit implicitement :

```
public class personne
    inherits object
```

La classe *object* définit une méthode virtuelle **Tostring** :



La méthode *ToString* rend le nom de la classe à laquelle appartient l'objet comme le montre l'exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test2
Sub Main()
    ' un enseignant
    Console.Out.WriteLine(New enseignant("Lucile", "Dumas", 56, 61).ToString())
    ' une personne
    Console.Out.WriteLine(New personne("Jean", "Dupont", 30).ToString())
End Sub
End Module
```

```
End Sub
End Module
```

Les résultats produits sont les suivants :

```
Construction personne(string, string, int)
Construction enseignant(string,string,int,int)
enseignant
Construction personne(string, string, int)
personne
```

On remarquera que bien que nous n'ayons pas redéfini la méthode *ToString* dans les classes *personne* et *enseignant*, on peut cependant constater que la méthode *ToString* de la classe *object* est quand même capable d'afficher le nom réel de la classe de l'objet. Redéfinissons la méthode *ToString* dans les classes *personne* et *enseignant* :

```
' ToString
Public Overrides Function ToString() As String
' on rend la propriété identite
Return identite
End Function
```

La définition est la même dans les deux classes. Considérons le programme de test suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
Sub Main()
' un enseignant
Dim e As New enseignant("Lucile", "Dumas", 56, 61)
affiche(e)
' une personne
Dim p As New personne("Jean", "Dupont", 30)
affiche(p)
End Sub

' affiche
Sub affiche(ByVal p As personne)
' affiche identité de p
Console.Out.WriteLine(p.identite)
End Sub
End Module
```

Les résultats d'exécution sont les suivants :

```
Construction personne(string, string, int)
Construction enseignant(string,string,int,int)
enseignant (personne (Lucile,Dumas,56),61)
Construction personne(string, string, int)
personne (Jean,Dupont,30)
```

## 2.3 Définir un indexeur pour une classe

Considérons la classe `[ArrayList]` prédéfinie dans la plate-forme .NET. Cette classe permet de mémoriser des objets dans une liste. Elle appartient à l'espace de noms `[System.Collections]`. Dans l'exemple suivant, nous utilisons cette classe pour mémoriser une liste de personnes (au sens large) :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections

' classe listeDePersonnes
Public Class listeDePersonnes
Inherits ArrayList

' pour ajouter une personne à la liste
```

```

Public Overloads Sub Add(ByVal p As personne)
    MyBase.Add(p)
End Sub

' un indexeur
Default Public Shadows Property Item(ByVal i As Integer) As personne
    Get
        Return CType(MyBase.Item(i), personne)
    End Get
    Set(ByVal Value As personne)
        MyBase.Item(i) = Value
    End Set
End Property

' un autre indexeur
Default Public Shadows ReadOnly Property Item(ByVal N As String) As Integer
    Get
        ' on recherche la personne de nom N
        Dim i As Integer
        For i = 0 To Count - 1
            If CType(Me(i), personne).nom = N Then
                Return i
            End If
        Next i
        Return -1
    End Get
End Property

' toString
Public Overrides Function ToString() As String
    ' rend (é11, é12, ..., éln)
    Dim liste As String = "("
    Dim i As Integer
    ' on parcourt le tableau dynamique
    For i = 0 To (Count - 2)
        liste += "[" + Me(i).ToString + "]" + ","
    Next i
    ' dernier élément
    If Count <> 0 Then
        liste += "[" + Me(i).ToString + "]"
    End If
    liste += ")"
    Return liste
End Function
End Class

```

Donnons quelques informations sur certaines propriétés et méthodes de la classe [ArrayList] :

Count	attribut donnant le nombre d'éléments de la liste
Add(Object)	méthode permettant d'ajouter un objet à la liste
Item(Integer i)	méthode donnant l'élément i de la liste

Nous remarquons que pour obtenir l'élément n° i de liste, nous n'avons pas écrit [liste.Item(i)] mais directement [liste(i)], ce qui à priori semble erroné. Ceci est néanmoins possible parce que la classe [ArrayList] définit une propriété par défaut [Item] selon la syntaxe analogue à la suivante :

```

Default Public Property Item(ByVal i As Integer) As Object
    Get
    ...
    End Get
    Set(ByVal Value As personne)
    ...
    End Set
End Property

```

Lorsque le compilateur rencontre la notation [liste(i)], il cherche si la classe [ArrayList] a défini une propriété avec la signature suivante :

```

Default Public Property Proc(ByVal var As Integer) As Type

```

Ici, il trouvera la procédure [Item]. Il traduira alors l'écriture [liste(i)] en [liste.Item(i)]. On appelle la propriété [Item], la propriété indexée par défaut de la classe [ArrayList]. L'exécution du programme précédent donne les résultats suivants :

```

dos>vbc /r:personne.dll /r:enseignant.dll lstpersonnes1.vb

```

```
dos>dir
11/03/2004 18:26          3 584 enseignant.dll
12/03/2004 15:34          3 584 lstpersonnes1.exe
12/03/2004 13:39          661 lstpersonnes1.vb
11/03/2004 18:26          4 096 personne.dll
```

```
dos>lstpersonnes1
Construction personne(string, string, int)
Construction personne(string, string, int)
Construction personne(string, string, int)
Construction enseignant(string,string,int,int)
personne (paul,chenou,31)
personne (nicole,chenou,11)
enseignant (personne (jacques, sileau,33) ,61)
```

Créons une classe appelé [listeDePersonnes] qui serait une liste de personnes, donc une liste particulière qu'il semble naturel de dériver de la classe [ArrayList] :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections

' classe listeDePersonnes
Public Class listeDePersonnes
    Inherits ArrayList

    ' pour ajouter une personne à la liste
    Public Shadows Sub Add(ByVal p As Object)
        ' il faut que p soit une personne
        If Not TypeOf (p) Is personne Then
            Throw New Exception("L'objet ajouté (" + p.ToString + ") n'est pas une personne")
        Else
            MyBase.Add(p)
        End If
    End Sub

    ' un indexeur
    Default Public Shadows Property Index(ByVal i As Integer) As personne
    Get
        Return CType(MyBase.Item(i), personne)
    End Get
    Set(ByVal Value As personne)
        MyBase.Item(i) = Value
    End Set
End Property

    ' toString
    Public Overrides Function ToString() As String
        ' rend (é11, é12, ..., é1n)
        Dim liste As String = "("
        Dim i As Integer
        ' on parcourt le tableau dynamique
        For i = 0 To (Count - 2)
            liste += "[" + Me(i).ToString + "]" + ","
        Next i
        ' dernier élément
        If Count <> 0 Then
            liste += "[" + Me(i).ToString + "]"
        End If
        liste += ")"
        Return liste
    End Function
End Class
```

La classe présente les méthodes et propriétés suivantes :

Tostring	rend une chaîne de caractères "représentant" le contenu de la liste
Add(personne)	méthode permettant d'ajouter une personne à la liste
Item(Integer i)	propriété indexée par défaut donnant la personne i de la liste

Considérons les points nouveaux :

Une nouvelle méthode [Add] est créée.

```
' pour ajouter une personne à la liste
Public Shadows Sub Add(ByVal p As Object)
' il faut que p soit une personne
If Not TypeOf (p) Is personne Then
    Throw New Exception("L'objet ajouté (" + p.ToString + ") n'est pas une personne")
Else
    MyBase.Add(p)
End If
End Sub
```

Il en existe déjà une dans la classe parent [ArrayList] avec la même signature, d'où le mot clé [Shadows] pour indiquer que la nouvelle procédure remplace celle de la classe parent. La méthode [Add] de la classe fille vérifie que l'objet ajouté est bien de type [personne] ou dérivé avec la fonction [TypeOf]. Si ce n'est pas le cas, une exception est lancée avec l'instruction [Throw]. Si l'objet ajouté est bien de type [personne], la méthode [Add] de la classe de base est utilisée pour faire l'ajout.

Une propriété indexée est créée :

```
' un indexeur
Default Public Shadows Property Index(ByVal i As Integer) As personne
Get
    Return CType(MyBase.Item(i), personne)
End Get
Set(ByVal Value As personne)
    MyBase.Item(i) = Value
End Set
End Property
```

Il existe déjà une propriété par défaut appelée [Item] dans la classe de base avec la même signature. Aussi est-on obligé d'utiliser le mot clé [Shadows] pour indiquer que la nouvelle propriété indexée [Index] va "cacher" celle de la classe de base [Item]. On notera que ceci est vrai même si les deux propriétés ne portent pas le même nom. La propriété [Index] permet de référencer la personne n° i de la liste. Elle s'appuie sur la propriété [Item] de la classe de base pour avoir accès à l'élément n° i de l'objet [ArrayList] sous-jacent. Des changements de type sont opérés pour tenir compte que la propriété [Item] travaille avec des éléments de type [Object] alors que la propriété [Index] travaille avec des éléments de type [personne].

Enfin, nous redéfinissons (Overrides) la méthode [ToString] de la classe [ArrayList] :

```
' toString
Public Overrides Function ToString() As String
' rend (é11, é12, ..., éln)
Dim liste As String = "("
Dim i As Integer
' on parcourt le tableau dynamique
For i = 0 To (Count - 2)
    liste += "[" + Me(i).ToString + "]" + ","
Next i
' dernier élément
If Count <> 0 Then
    liste += "[" + Me(i).ToString + "]"
End If
liste += ")"
Return liste
End Function
```

Cette méthode rend une chaîne de caractères de la forme "(e1,e2,...,en)" où ei sont les éléments de la liste. On remarquera la notation [Me(i)] qui désigne l'élément n° i de l'objet courant [Me]. C'est la propriété indexée par défaut qui est utilisée. Ainsi [Me(i)] est équivalent à [Me.Index(i)].

Le code de la classe est placé dans le fichier [lstpersonnes2.vb] et compilé :

```
dos>dir
11/03/2004  18:26                3 584 enseignant.dll
12/03/2004  15:45                970 lstpersonnes2.vb
11/03/2004  18:26                4 096 personne.dll
```

```
dos>vbc /r:personne.dll /r:enseignant.dll /t:library lstpersonnes2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version 1.1.4322.573
```

```
dos>dir
11/03/2004 18:26          3 584 enseignant.dll
12/03/2004 15:50          3 584 lstpersonnes2.dll
12/03/2004 15:45          970 lstpersonnes2.vb
11/03/2004 18:26          4 096 personne.dll
```

Un programme de test est construit :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Collections

Module test
Sub Main()
    ' création d'une liste vide de personnes
    Dim liste As listeDePersonnes = New listeDePersonnes
    ' création de personnes
    Dim p1 As personne = New personne("paul", "chenou", 31)
    Dim p2 As personne = New personne("nicole", "chenou", 11)
    Dim e1 As enseignant = New enseignant("jacques", "sileau", 33, 61)
    ' remplissage de la liste
    liste.Add(p1)
    liste.Add(p2)
    liste.Add(e1)
    ' affichage de la liste
    Console.Out.WriteLine(liste.ToString)
    ' ajout d'un objet différent de personne
    Try
        liste.Add(4)
    Catch e As Exception
        Console.Error.WriteLine(e.Message)
    End Try
End Sub
End Module
```

Il est compilé :

```
dos>vbc /r:personne.dll /r:enseignant.dll /r:lstpersonnes2.dll test.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573
```

```
dos>dir
11/03/2004 18:26          3 584 enseignant.dll
12/03/2004 15:50          3 584 lstpersonnes2.dll
12/03/2004 15:45          970 lstpersonnes2.vb
11/03/2004 18:26          4 096 personne.dll
12/03/2004 15:50          3 584 test.exe
12/03/2004 15:49          623 test.vb
```

puis exécuté :

```
dos>test
Construction personne(string, string, int)
Construction personne(string, string, int)
Construction personne(string, string, int)
Construction enseignant(string, string, int, int)
([personne(paul, chenou, 31)], [personne(nicole, chenou, 11)], [enseignant(personne(jacques, sileau, 33), 61)])
L'objet ajouté (4) n'est pas une personne
```

On pourrait vouloir écrire

```
dim p as personne=1("nom")
```

où l serait de type [listeDePersonnes]. Ici, on veut indexer la liste l non plus par un n° d'élément mais par un nom de personne. Pour cela on définit une nouvelle propriété indexée par défaut :

```
' un autre indexeur
Default Public Shadows ReadOnly Property Item(ByVal N As String) As Integer
Get
    ' on recherche la personne de nom N
    Dim i As Integer
    For i = 0 To Count - 1
        If CType(Me(i), personne).nom = N Then
```

```

        Return i
    End If
Next i
Return -1
End Get
End Property

```

La première ligne

```
Default Public Shadows ReadOnly Property Index(ByVal N As String) As Integer
```

indique qu'on crée de nouveau une propriété indexée par défaut. Toutes les propriétés par défaut doivent porter le même nom, ici [Index]. La nouvelle propriété [Index] indexe la classe *listeDePersonnes* par une chaîne de caractères N. Le résultat de *listeDePersonnes(N)* est un entier. Cet entier sera la position dans la liste, de la personne portant le nom N ou -1 si cette personne n'est pas dans la liste. On ne définit que la propriété *get* interdisant ainsi l'écriture *listeDePersonnes("nom")=valeur* qui aurait nécessité la définition de la propriété *set*. D'où le mot clé [ReadOnly]. Le mot clé [Shadows] est nécessaire pour cacher la propriété par défaut de la classe de base (bien qu'elle n'ait pas la même signature).

Dans le corps du *get*, on parcourt la liste des personnes à la recherche du nom N passé en paramètre. Si on le trouve en position i, on renvoie i sinon on renvoie -1.

Un nouveau programme de test pourrait être le suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' pg de test
Module test
Sub Main()
    ' une liste de personnes
    Dim l As New listeDePersonnes
    ' ajout de personnes
    l.Add(New personne("jean", "dumornet", 10))
    l.Add(New personne("pauline", "duchemin", 12))
    ' affichage
    Console.Out.WriteLine("l=" + l.ToString())
    l.Add(New personne("jacques", "tartifume", 27))
    Console.Out.WriteLine("l=" + l.ToString())
    ' changement élément 1
    l(1) = New personne("sylvie", "cachan", 5)
    ' affichage élément 1
    Console.Out.WriteLine("l[1]=" + l(1).ToString())
    ' affichage liste l
    Console.Out.WriteLine("l=" + l.ToString())
    ' recherche de personnes
    Dim noms() As String = New [String]() {"cachan", "inconnu"}
    Dim i As Integer
    For i = 0 To noms.Length - 1
        Dim inom As Integer = l(noms(i))
        If inom <> -1 Then
            Console.Out.WriteLine("personne(" & noms(i) & ")=" & l(inom).ToString())
        Else
            Console.Out.WriteLine("personne(" + noms(i) + ") n'existe pas")
        End If
    Next i
End Sub
End Module

```

L'exécution donne les résultats suivants :

```

personne(string, string, int)
Construction personne(string, string, int)
l=([personne(jean,dumornet,10)], [personne(pauline,duchemin,12)])
Construction personne(string, string, int)
l=( [personne(jean,dumornet,10)], [personne(pauline,duchemin,12)], [personne(jacques,tartifume,27)])
Construction personne(string, string, int)
l[1]=personne(sylvie,cachan,5)
l=( [personne(jean,dumornet,10)], [personne(sylvie,cachan,5)], [personne(jacques,tartifume,27)])
personne(cachan)=personne(sylvie,cachan,5)
personne(inconnu) n'existe pas

```

## 2.4 Les structures

La structure VB.NET est directement issue de la structure du langage C et est très proche de la classe. Une structure est définie comme suit :

```
Structure spersonne
' attributs
'...
' propriétés
'...
' constructeurs
'...
' méthodes
End Structure
```

Il y a, malgré une similitude de déclaration, des différences importantes entre **classe** et **structure**. La notion d'héritage n'existe par exemple pas avec les structures. Si on écrit une classe qui ne doit pas être dérivée, quelles sont les différences entre structure et classe qui vont nous aider à choisir entre les deux ? Aidons-nous de l'exemple suivant pour le découvrir :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' structure spersonne
Structure spersonne
    Public nom As String
    Public age As Integer
End Structure

' classe cpersonne
Class cpersonne
    Public nom As String
    Public age As Integer
End Class

' une module de test
Public Module test
    Sub Main()
        ' une spersonne p1
        Dim sp1 As spersonne
        sp1.nom = "paul"
        sp1.age = 10
        Console.Out.WriteLine("sp1=spersonne(" & sp1.nom & "," & sp1.age & ")")
        ' une spersonne p2
        Dim sp2 As spersonne = sp1
        Console.Out.WriteLine("sp2=spersonne(" & sp2.nom & "," & sp2.age & ")")
        ' sp2 est modifié
        sp2.nom = "nicole"
        sp2.age = 30
        ' vérification sp1 et sp2
        Console.Out.WriteLine("sp1=cpersonne(" & sp1.nom & "," & sp1.age & ")")
        Console.Out.WriteLine("sp2=cpersonne(" & sp2.nom & "," & sp2.age & ")")

        ' une cpersonne cp1
        Dim cp1 As New cpersonne
        cp1.nom = "paul"
        cp1.age = 10
        Console.Out.WriteLine("cP1=cpersonne(" & cp1.nom & "," & cp1.age & ")")
        ' une cpersonne P2
        Dim cp2 As cpersonne = cp1
        Console.Out.WriteLine("cP2=cpersonne(" & cp2.nom & "," & cp2.age & ")")
        ' P2 est modifié
        cp2.nom = "nicole"
        cp2.age = 30
        ' vérification P1 et P2
        Console.Out.WriteLine("cP1=cpersonne(" & cp1.nom & "," & cp1.age & ")")
        Console.Out.WriteLine("cP2=cpersonne(" & cp2.nom & "," & cp2.age & ")")
    End Sub
End Module
```

Si on exécute ce programme, on obtient les résultats suivants :

```
sp1=spersonne (paul,10)
sp2=spersonne (paul,10)
sp1=cpersonne (paul,10)
```

```

sp2=cpersonne (nicole,30)
cP1=cpersonne (paul,10)
cP2=cpersonne (paul,10)
cP1=cpersonne (nicole,30)
cP2=cpersonne (nicole,30)

```

Là où dans les pages précédentes de ce chapitre on utilisait une classe *personne*, nous utilisons maintenant une structure *spersonne* :

```

' structure spersonne
Structure spersonne
  Public nom As String
  Public age As Integer
End Structure

```

La déclaration

```
Dim sp1 As spersonne
```

crée une structure (nom,age) et la valeur de sp1 est cette structure elle-même.

La déclaration

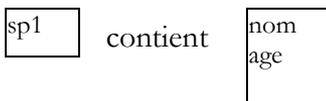
```
Dim cp1 As New cpersonne
```

crée un objet [cpersonne] (grosso modo l'équivalent de notre structure) et *cp1* est alors l'adresse (la référence) de cet objet.

Résumons

- dans le cas de la structure, la **valeur** de sp1 est la structure elle-même
- dans le cas de la classe, la **valeur** de p1 est l'**adresse** de l'objet créé

Structure p1



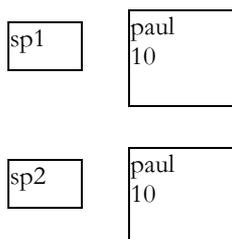
Objet p1



Lorsque dans le programme on écrit

```
Dim sp2 As spersonne = sp1
```

une nouvelle structure (nom,age) est créée et initialisée avec la valeur de *p1* donc la structure elle-même.

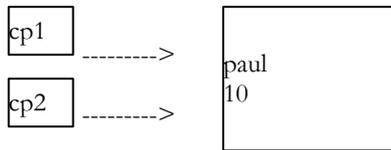


La structure de sp1 est donc **dupliquée** dans sp2. C'est une recopie de valeur.

L'instruction

```
Dim cp2 As cpersonne = cp1
```

agit différemment. La valeur de cp1 est recopiée dans cp2, mais comme cette valeur est en fait l'adresse de l'objet, celui-ci n'est pas dupliqué. Il a simplement deux références sur lui :



Dans le cas de la structure, si on modifie la valeur de *sp2* on ne modifie pas la valeur de *sp1*, ce que montre le programme. Dans le cas de l'objet, si on modifie l'objet pointé par *cp2*, celui pointé par *cp1* est modifié puisque c'est le même. C'est ce que montrent également les résultats du programme.

On retiendra donc de ces explications que :

- la valeur d'une variable de type structure est la structure elle-même
- la valeur d'une variable de type objet est l'adresse de l'objet pointé

Une fois cette différence fondamentale comprise, la structure se montre très proche de la classe comme le montre le nouvel exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' structure personne
Structure personne
' attributs
Private _nom As String
Private _age As Integer

' propriétés
Public Property nom() As String
Get
    Return _nom
End Get
Set(ByVal Value As String)
    _nom = value
End Set
End Property

Public Property age() As Integer
Get
    Return _age
End Get
Set(ByVal Value As Integer)
    _age = value
End Set
End Property

' Constructeur
Public Sub New(ByVal NOM As String, ByVal AGE As Integer)
    _nom = NOM
    _age = AGE
End Sub'New

' TOSTRING
Public Overrides Function ToString() As String
    Return "personne(" & nom & "," & age & ")"
End Function
End Structure

' un module de test
Module test
Sub Main()
    ' une personne p1
    Dim p1 As New personne("paul", 10)
    Console.Out.WriteLine("p1=" & p1.ToString())
    ' une personne p2
    Dim p2 As personne = p1
    Console.Out.WriteLine("p2=" & p2.ToString())
    ' p2 est modifié
    p2.nom = "nicole"
    p2.age = 30
    ' vérification p1 et p2
    Console.Out.WriteLine("p1=" & p1.ToString())
    Console.Out.WriteLine("p2=" & p2.ToString())
End Sub
```

On obtient les résultats d'exécution suivants :

```
p1=personne (paul,10)
p2=personne (paul,10)
p1=personne (paul,10)
p2=personne (nicole,30)
```

La seule notable différence ici entre structure et classe, c'est qu'avec une classe, les objets p1 et p2 auraient eu la même valeur à la fin du programme, celle de p2.

## 2.5 Les interfaces

Une interface est un ensemble de prototypes de méthodes ou de propriétés qui forme un contrat. Une classe qui décide d'implémenter une interface s'engage à fournir une implémentation de toutes les méthodes définies dans l'interface. C'est le compilateur qui vérifie cette implémentation. Voici par exemple la définition de l'interface *Istats* :

```
Public Interface Istats
    Function moyenne() As Double
    Function écartType() As Double
End Interface
```

Toute classe implémentant cette interface sera déclarée comme

```
public class C
    Implements Istats
    ...
    function moyenne() as Double Implements Istats.moyenne
    ...
    end function
    function écartType () as Double Implements Istats.écartType
    ...
    end function
end class
```

Les méthodes *[moyenne]* et *[écartType]* devront être définies dans la classe C. Considérons le code suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' structure
Public Structure élève
    Public _nom As String
    Public _note As Double

' constructeur
Public Sub New(ByVal NOM As String, ByVal NOTE As Double)
    Me._nom = NOM
    Me._note = NOTE
End Sub
End Structure

' classe notes
Public Class notes
    ' attribut
    Protected _matière As String
    Protected _élèves() As élève

' constructeur
Public Sub New(ByVal MATIERE As String, ByVal ELEVES() As élève)
    ' mémorisation élèves & matière
    Me._matière = MATIERE
    Me._élèves = ELEVES
End Sub

' ToString
Public Overrides Function ToString() As String
    Dim valeur As String = "matière=" + _matière + ", notes=("
    Dim i As Integer
    ' on concatène toutes les notes
    For i = 0 To (_élèves.Length - 1) - 1
```

```

    valeur &= "[" & _élèves(i)._nom & "," & _élèves(i)._note & "], "
Next i
'dernière note
If _élèves.Length <> 0 Then
    valeur &= "[" & _élèves(i)._nom & "," & _élèves(i)._note & "]"
End If
valeur += ")"
' fin
Return valeur
End Function
End Class

```

La classe *notes* rassemble les notes d'une classe dans une matière :

```

Public Class notes
    ' attribut
    Protected _matière As String
    Protected _élèves() As élève

```

Les attributs sont déclarés *protected* pour être accessibles d'une classe dérivée. Le type *élève* est une structure mémorisant le nom de l'élève et sa note dans la matière :

```

Public Structure élève
    Public _nom As String
    Public _note As Double

    ' constructeur
    Public Sub New(ByVal NOM As String, ByVal NOTE As Double)
        Me._nom = NOM
        Me._note = NOTE
    End Sub
End Structure

```

Nous décidons de dériver cette classe *notes* dans une classe *notesStats* qui aurait deux attributs supplémentaires, la moyenne et l'écart-type des notes :

```

Public Class notesStats
    Inherits notes
    Implements Istats

    ' attributs
    Private _moyenne As Double
    Private _écartType As Double

```

La classe *notesStats* implémente l'interface *Istats* suivante :

```

Public Interface Istats
    Function moyenne() As Double
    Function écartType() As Double
End Interface

```

Cela signifie que la classe *notesStats* doit avoir deux méthodes appelées *moyenne* et *écartType* avec la signature indiquée dans l'interface *Istats*. La classe *notesStats* est la suivante :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Class notesStats
    Inherits notes
    Implements Istats

    ' attributs
    Private _moyenne As Double
    Private _écartType As Double

    ' constructeur
    Public Sub New(ByVal MATIERE As String, ByVal ELEVES() As élève)
        MyBase.New(MATIERE, ELEVES)
        ' calcul moyenne des notes
        Dim somme As Double = 0
        Dim i As Integer
        For i = 0 To ELEVES.Length - 1
            somme += ELEVES(i)._note
        Next i

```

```

If ELEVES.Length <> 0 Then
    _moyenne = somme / ELEVES.Length
Else
    _moyenne = -1
End If
End If
' écart-type
Dim carrés As Double = 0
For i = 0 To ELEVES.Length - 1
    carrés += Math.Pow(ELEVES(i)._note - _moyenne, 2)
Next i
If ELEVES.Length <> 0 Then
    _écartType = Math.Sqrt((carrés / ELEVES.Length))
Else
    _écartType = -1
End If
End Sub

' ToString
Public Overrides Function ToString() As String
    Return MyBase.ToString() & ",moyenne=" & _moyenne & ",écart-type=" & _écartType
End Function 'ToString

' méthodes de l'interface Istats
Public Function moyenne() As Double Implements Istats.moyenne
    ' rend la moyenne des notes
    Return _moyenne
End Function

Public Function écartType() As Double Implements Istats.écartType
    ' rend l'écart-type
    Return _écartType
End Function
End Class

```

La moyenne *\_moyenne* et l'écart-type *\_écartType* sont calculés dès la construction de l'objet. Aussi les méthodes *moyenne* et *écartType* n'ont-elles qu'à rendre la valeur des attributs *\_moyenne* et *\_écartType*. Les deux méthodes rendent -1 si le tableau des élèves est vide.

La classe de test suivante :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
    Sub Main()
        ' qqs élèves & notes
        Dim ELEVES() As élève = {New élève("paul", 14), New élève("nicole", 16), New élève("jacques", 18)}
        ' qu'on enregistre dans un objet notes
        Dim anglais As New notes("anglais", ELEVES)
        ' et qu'on affiche
        Console.Out.WriteLine((anglais.ToString))
        ' idem avec moyenne et écart-type
        anglais = New notesStats("anglais", ELEVES)
        Console.Out.WriteLine((anglais.ToString))
    End Sub
End Module

```

donne les résultats :

```

matière=anglais, notes=([paul,14],[nicole,16],[jacques,18])
matière=anglais, notes=([paul,14],[nicole,16],[jacques,18]),moyenne=16,écart-type=1,63299316185545

```

La classe *notesStats* aurait très bien pu implémenter les méthodes *moyenne* et *écartType* pour elle-même sans indiquer qu'elle implémentait l'interface *Istats*. Quel est l'intérêt des interfaces ? C'est le suivant : une fonction peut admettre pour paramètre formel une donnée ayant le type d'une interface I. Tout objet d'une classe C implémentant l'interface I pourra alors être paramètre effectif de cette fonction. Considérons l'exemple suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' une interface Iexemple

```

```

Public Interface Iexemple
    Function ajouter(ByVal i As Integer, ByVal j As Integer) As Integer
    Function soustraire(ByVal i As Integer, ByVal j As Integer) As Integer
End Interface

' une 1ère classe
Public Class classe1
    Implements Iexemple

    Public Function ajouter(ByVal a As Integer, ByVal b As Integer) As Integer Implements Iexemple.ajouter
        Return a + b + 10
    End Function

    Public Function soustraire(ByVal a As Integer, ByVal b As Integer) As Integer Implements
Iexemple.soustraire
        Return a - b + 20
    End Function
End Class

' une 2ième classe
Public Class classe2
    Implements Iexemple

    Public Function ajouter(ByVal a As Integer, ByVal b As Integer) As Integer Implements Iexemple.ajouter
        Return a + b + 100
    End Function

    Public Function soustraire(ByVal a As Integer, ByVal b As Integer) As Integer Implements
Iexemple.soustraire
        Return a - b + 200
    End Function
End Class

```

L'interface *Iexemple* définit deux méthodes *ajouter* et *soustraire*. Les classes *classe1* et *classe2* implémentent cette interface. On remarquera que ces classes ne font rien d'autre, ceci par souci de simplification de l'exemple. Maintenant considérons l'exemple suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

' classe de test
Module test

    ' calculer
    Sub calculer(ByVal i As Integer, ByVal j As Integer, ByVal inter As Iexemple)
        Console.Out.WriteLine(inter.ajouter(i, j))
        Console.Out.WriteLine(inter.soustraire(i, j))
    End Sub

    ' la fonction Main
    Sub Main()
        ' création de deux objets classe1 et classe2
        Dim c1 As New classe1
        Dim c2 As New classe2
        ' appels de la fonction statique calculer
        calculer(4, 3, c1)
        calculer(14, 13, c2)
    End Sub
End Module

```

La fonction *calculer* admet pour paramètre un élément de type *Iexemple*. Elle pourra donc recevoir pour ce paramètre aussi bien un objet de type *classe1* que de type *classe2*. C'est ce qui est fait dans la procédure *Main* avec les résultats suivants :

```

17
21
127
201

```

On voit donc qu'on a là une propriété proche du polymorphisme vu pour les classes. Si un ensemble de classes **Ci** non liées entre-elles par héritage (donc on ne peut utiliser le polymorphisme de l'héritage) présente un ensemble de méthodes de même signature, il peut être intéressant de regrouper ces méthodes dans une interface **I** dont hériteraient toutes les classes concernées. Des instances de ces classes **Ci** peuvent alors être utilisées comme paramètres de fonctions admettant un paramètre de type **I**, c.a.d. des fonctions n'utilisant que les méthodes des objets **Ci** définies dans l'interface **I** et non les attributs et méthodes particuliers des différentes classes **Ci**. Notons enfin que l'héritage d'interfaces peut être multiple, c.a.d. qu'on peut écrire

```
Public Class classe
    Implements I1, I2, ...
```

où les *I* sont des interfaces.

## 2.6 Les espaces de noms

Pour écrire une ligne à l'écran, nous utilisons l'instruction

```
Console.Out.WriteLine(...)
```

Si nous regardons la définition de la classe *Console*

```
Namespace: System
Assembly: Mscorlib (in Mscorlib.dll)
```

on découvre qu'elle fait partie de l'espace de noms *System*. Cela signifie que la classe *Console* devrait être désignée par *System.Console* et on devrait en fait écrire :

```
System.Console.Out.WriteLine(...)
```

On évite cela en utilisant une clause *imports* :

```
imports System
...
Console.Out.WriteLine(...)
```

On dit qu'on **importe** l'espace de noms *System* avec la clause *imports*. Lorsque le compilateur va rencontrer le nom d'une classe (ici *Console*) il va chercher à la trouver dans les différents espaces de noms importés par les clauses *imports*. Ici il trouvera la classe *Console* dans l'espace de noms *System*. Notons maintenant la seconde information attachée à la classe *Console* :

```
Assembly: Mscorlib (in Mscorlib.dll)
```

Cette ligne indique dans quelle "assemblage" se trouve la définition de la classe *Console*. Lorsqu'on compile en-dehors de Visual Studio.NET et qu'on doit donner les références des différentes *dll* contenant les classes que l'on doit utiliser cette information peut s'avérer utile. On rappelle que pour référencer les *dll* nécessaires à la compilation d'une classe, on écrit :

```
vbc /r:fic1.dll /r:fic2.dll ... prog.vb
```

Lorsqu'on crée une classe, on peut la créer à l'intérieur d'un espace de noms. Le but de ces espaces de noms est d'éviter les conflits de noms entre classes lorsque celles-ci sont vendues par exemple. Considérons deux entreprises E1 et E2 distribuant des classes empaquetées respectivement dans les *dll*, *E1.dll* et *E2.dll*. Soit un client C qui achète ces deux ensembles de classes dans lesquelles les deux entreprises ont défini toutes deux une classe *personne*. Le client C compile un programme de la façon suivante :

```
vbc /r:E1.dll /r:E2.dll prog.vb
```

Si le source *prog.vb* utilise la classe *personne*, le compilateur ne saura pas s'il doit prendre la classe *personne* de *E1.dll* ou celle de *E2.dll*. Il signalera une erreur. Si l'entreprise E1 prend soin de créer ses classes dans un espace de noms appelé E1 et l'entreprise E2 dans un espace de noms appelé E2, les deux classes *personne* s'appelleront alors *E1.personne* et *E2.personne*. Le client devra employer dans ses classes soit *E1.personne*, soit *E2.personne* mais pas *personne*. L'espace de noms permet de lever l'ambiguïté. Pour créer une classe dans un espace de noms, on écrit :

```
Namespace istia.st
    Public Class personne
        ' définition de la classe
        ...
    end class
end namespace
```

Pour l'exemple, créons dans un espace de noms notre classe *personne* étudiée précédemment. Nous choisirons *istia.st* comme espace de noms. La classe *personne* devient :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
```

```

' création de l'espace de nom istia.st
Namespace istia.st
Public Class personne
    ' attributs
    Private prenom As String
    Private nom As String
    Private age As Integer

    ' méthode
    Public Sub initialise(ByVal P As String, ByVal N As String, ByVal age As Integer)
        Me.prenom = P
        Me.nom = N
        Me.age = age
    End Sub

    ' méthode
    Public Sub identifie()
        Console.Out.WriteLine((prenom & "," & nom & "," & age))
    End Sub
End Class
End Namespace

```

Cette classe est compilée dans *personne.dll* :

```

dos>dir
11/03/2004  18:27                610 personne.vb

dos>vbc /t:library personne.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>dir
12/03/2004  18:06                3 584 personne.dll
11/03/2004  18:27                610 personne.vb

```

Maintenant utilisons la classe *personne* dans une classe de test :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports istia.st

' pg test
Public Module test
    Sub Main()
        Dim p1 As New personne
        p1.initialise("Jean", "Dupont", 30)
        p1.identifie()
    End Sub
End Module

```

Pour éviter d'écrire

```
Dim p1 As New istia.st.personne
```

nous avons importé l'espace de noms *istia.st* avec une clause *imports* :

```
Imports istia.st
```

Maintenant compilons le programme de test :

```

dos>dir
12/03/2004  18:06                3 584 personne.dll
11/03/2004  18:27                610 personne.vb
12/03/2004  18:05                254 test.vb

dos>vbc /r:personne.dll test.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>dir
12/03/2004  18:06                3 584 personne.dll
11/03/2004  18:27                610 personne.vb

```

Cela produit un fichier *test.exe* qui exécuté donne les résultats suivants :

Jean, Dupont, 30

## 2.7 L'exemple IMPOTS

On reprend le calcul de l'impôt déjà étudié dans le chapitre précédent et on le traite en utilisant une classe. Rappelons le problème :

On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer :

- on calcule le nombre de parts du salarié  $\text{nbParts} = \text{nbEnfants} / 2 + 1$  s'il n'est pas marié,  $\text{nbEnfants} / 2 + 2$  s'il est marié, où *nbEnfants* est son nombre d'enfants.
- s'il a au moins trois enfants, il a une demie part de plus
- on calcule son revenu imposable  $R = 0.72 * S$  où *S* est son salaire annuel
- on calcule son coefficient familial  $QF = R / \text{nbParts}$
- on calcule son impôt *I*. Considérons le tableau suivant :

12620.0	0	0
13190	0.05	631
15640	0.1	1290.5
24740	0.15	2072.5
31810	0.2	3309.5
39970	0.25	4900
48360	0.3	6898.5
55790	0.35	9316.5
92970	0.4	12106
127860	0.45	16754.5
151250	0.50	23147.5
172040	0.55	30710
195000	0.60	39312
0	0.65	49062

Chaque ligne a 3 champs. Pour calculer l'impôt *I*, on recherche la première ligne où  $QF \leq \text{champ1}$ . Par exemple, si  $QF = 23000$  on trouvera la ligne

24740 0.15 2072.5

L'impôt *I* est alors égal à  $0.15 * R - 2072.5 * \text{nbParts}$ . Si  $QF$  est tel que la relation  $QF \leq \text{champ1}$  n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

0 0.65 49062

ce qui donne l'impôt  $I = 0.65 * R - 49062 * \text{nbParts}$ .

La classe **impot** sera définie comme suit :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Class impot
    ' les données nécessaires au calcul de l'impôt
    ' proviennent d'une source extérieure
    Private limites() , coeffR() , coeffN() As Decimal

    ' constructeur
    Public Sub New(ByVal LIMITES() As Decimal, ByVal COEFFR() As Decimal, ByVal COEFFN() As Decimal)
        ' on vérifie que les 3 tableaux ont la même taille
        Dim OK As Boolean = LIMITES.Length = COEFFR.Length And LIMITES.Length = COEFFN.Length
        If Not OK Then
            Throw New Exception("Les 3 tableaux fournis n'ont pas la même taille(" & LIMITES.Length & "," &
                COEFFR.Length & "," & COEFFN.Length & ")")
        End If
        ' c'est bon
        Me.limites = LIMITES
        Me.coeffR = COEFFR
        Me.coeffN = COEFFN
    End Sub
End Class
```

```

' calcul de l'impôt
Public Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Long) As Long
' calcul du nombre de parts
Dim nbParts As Decimal
If marié Then
    nbParts = CDec(nbEnfants) / 2 + 2
Else
    nbParts = CDec(nbEnfants) / 2 + 1
End If
If nbEnfants >= 3 Then
    nbParts += 0.5D
End If
' calcul revenu imposable & Quotient familial
Dim revenu As Decimal = 0.72D * salaire
Dim QF As Decimal = revenu / nbParts
' calcul de l'impôt
limites((limites.Length - 1)) = QF + 1
Dim i As Integer = 0
While QF > limites(i)
    i += 1
End While
Return CLng(revenu * coeffR(i) - nbParts * coeffN(i))
End Function
End Class

```

Un objet **impôt** est créé avec les données permettant le calcul de l'impôt d'un contribuable. C'est la partie stable de l'objet. Une fois cet objet créé, on peut appeler de façon répétée sa méthode **calculer** qui calcule l'impôt du contribuable à partir de son statut marital (marié ou non), son nombre d'enfants et son salaire annuel. Un programme de test pourrait être le suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

Module test
Sub Main()
' programme interactif de calcul d'impôt
' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
' le programme affiche alors l'impôt à payer
Const syntaxe As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire : salaire annuel en F"

' tableaux de données nécessaires au calcul de l'impôt
Dim limites() As Decimal = {12620D, 13190D, 15640D, 24740D, 31810D, 39970D, 48360D, 55790D, 92970D, 127860D, 151250D, 172040D, 195000D, 0D}
Dim coeffR() As Decimal = {0D, 0.05D, 0.1D, 0.15D, 0.2D, 0.25D, 0.3D, 0.35D, 0.4D, 0.45D, 0.5D, 0.55D, 0.6D, 0.65D}
Dim coeffN() As Decimal = {0D, 631D, 1290.5D, 2072.5D, 3309.5D, 4900D, 6898.5D, 9316.5D, 12106D, 16754.5D, 23147.5D, 30710D, 39312D, 49062D}

' création d'un objet impôt
Dim objImpôt As impot = Nothing
Try
    objImpôt = New impot(limites, coeffR, coeffN)
Catch ex As Exception
    Console.Error.WriteLine("L'erreur suivante s'est produite : " + ex.Message)
    Environment.Exit(1)
End Try
' boucle infinie
Dim marié As String
Dim nbEnfants As Integer
Dim salaire As Long
While True
' on demande les paramètres du calcul de l'impôt
    Console.Out.Write("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :")
    Dim paramètres As String = Console.In.ReadLine().Trim()
' qq chose à faire ?
    If paramètres Is Nothing OrElse paramètres = "" Then
        Exit While
    End If
' vérification du nombre d'arguments dans la ligne saisie
    Dim erreur As Boolean = False
    Dim args As String() = paramètres.Split(Nothing)
    Dim nbParamètres As Integer = args.Length

```

```

If nbParamètres <> 3 Then
    Console.Error.WriteLine(syntaxe)
    erreur = True
End If
' vérification de la validité des paramètres
If Not erreur Then
    ' marié
    marié = args(0).ToLower()
    If marié <> "o" And marié <> "n" Then
        erreur = True
    End If
    ' nbEnfants
    Try
        nbEnfants = Integer.Parse(args(1))
        If nbEnfants < 0 Then
            Throw New Exception
        End If
    Catch
        erreur = True
    End Try
    ' salaire
    Try
        salaire = Integer.Parse(args(2))
        If salaire < 0 Then
            Throw New Exception
        End If
    Catch
        erreur = True
    End Try
End If
' si les paramètres sont corrects - on calcule l'impôt
If Not erreur Then
    Console.Out.WriteLine(("impôt=" & objImpôt.calculer(marié = "o", nbEnfants, salaire) & " F"))
Else
    Console.Error.WriteLine(syntaxe)
End If
End While
End Sub
End Module

```

Voici un exemple d'exécution du programme précédent :

```

dir>dir
12/03/2004 18:20          1 483 impots.vb
12/03/2004 18:21          2 805 test.vb

```

```

dos>vbc /t:library impots.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

```

```

dir>dir
12/03/2004 18:24          4 096 impots.dll
12/03/2004 18:20          1 483 impots.vb
12/03/2004 18:21          2 805 test.vb

```

```

dos>vbc /r:impots.dll test.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

```

```

dos>dir
12/03/2004 18:24          4 096 impots.dll
12/03/2004 18:20          1 483 impots.vb
12/03/2004 18:26          6 144 test.exe
12/03/2004 18:21          2 805 test.vb

```

```

dos>test
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :x x x
syntaxe : marié nbEnfants salaire
marié : o pour marié, n pour non marié
nbEnfants : nombre d'enfants
salaire : salaire annuel en F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 200000
impôt=22504 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :

```

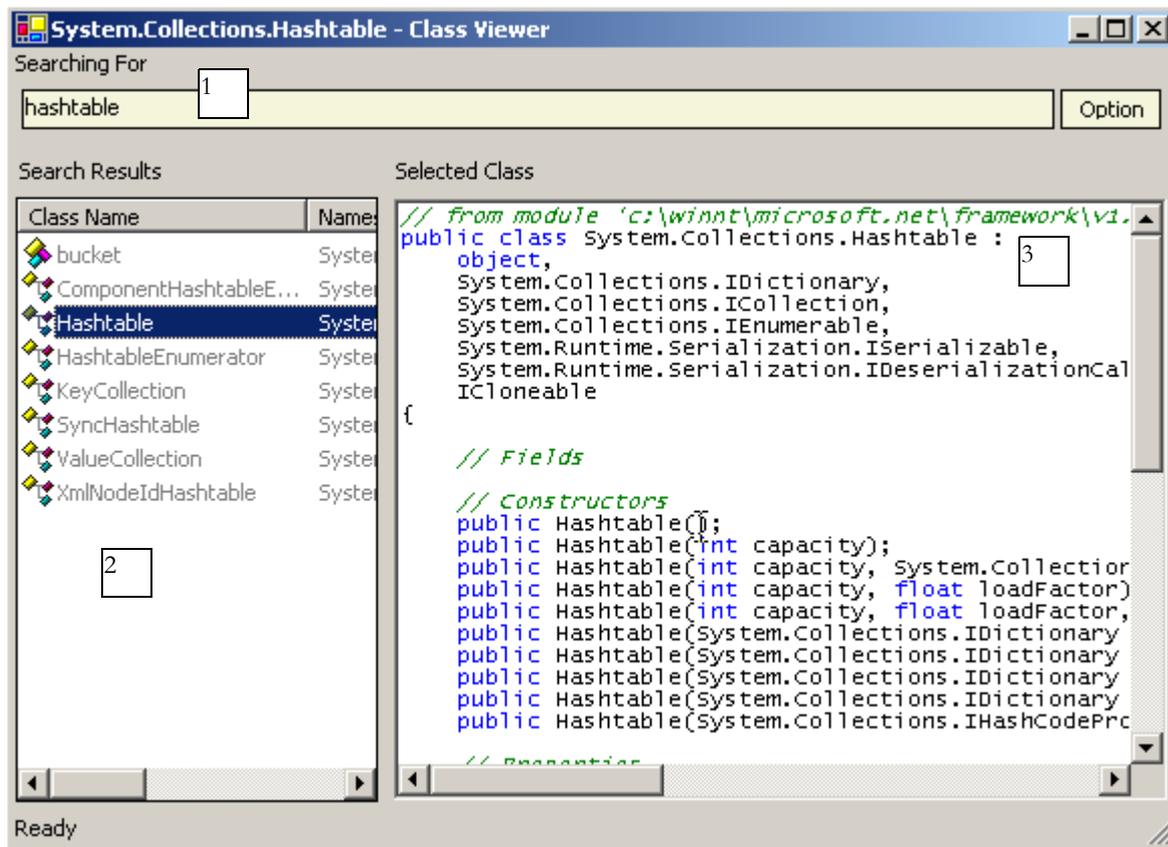
# 3. Classes .NET d'usage courant

Nous présentons ici quelques classes de la plate-forme .NET présentant un intérêt, même pour un débutant. Nous montrons tout d'abord comment obtenir des renseignements sur les centaines de classes disponibles.

## 3.1 Chercher de l'aide avec SDK.NET

### 3.1.1 wincv

Si on a installé seulement le SDK et pas Visual Studio.NET on pourra utiliser le programme *wincv.exe* situé normalement dans l'arborescence du sdk, par exemple *C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1*. Lorsqu'on lance cet utilitaire, on a l'interface suivante :



On tape en (1) le nom de la classe désirée. Cela ramène en (2) divers thèmes possibles. On choisit celui qui convient et on a le résultat en (3), ici la classe *HashTable*. Cette méthode convient si on connaît le nom de la classe que l'on cherche. Si on veut explorer la liste des possibilités offertes par la plate-forme .NET ou pourra utiliser le fichier HTML *StartHere.htm* situé lui aussi directement dans le dossier d'installation de SSK.Net, par exemple *C:\Program Files\Microsoft Visual Studio .NET 2003\SDK\v1.1*.

# Kit de développement Microsoft .NET Framework SDK

Bienvenue sur le site du Kit de développement Microsoft .NET Framework SDK. Ce Kit de développement .NET Framework SDK est la référence fondamentale pour les développeurs qui utilisent les technologies du .NET Framework.

## Mise en route

Pour les personnes qui ne connaissent pas les technologies .NET Framework, la section [Mise en route de .NET Framework](#) de la documentation du Kit de développement .NET Framework SDK est conçue pour vous guider.

## Documentation

La [documentation du Kit de développement .NET Framework SDK](#) fournit de nombreuses vues d'ensemble, tâches de programmation et informations de référence sur la bibliothèque de classes qui sont conçues pour vous aider à concevoir des applications efficaces, puissantes et dimensionnables basées sur les technologies .NET Framework.

Le [Tool Developer's Guide](#) apporte des informations utiles aux développeurs souhaitant créer des outils de développement de bas niveau qui fonctionnent à l'intérieur du .NET Framework, tels que des compilateurs, des navigateurs, des générateurs de profils et des débogueurs.

Pour obtenir une liste des modifications apportées à la bibliothèque de classes pour le .NET Framework version 1.1, consultez [Class Library Changes and Additions](#).

## Démarrages rapides, didacticiels et exemples

Les [Démarrages rapides, didacticiels et exemples](#) du Kit de développement .NET Framework SDK sont conçus pour vous familiariser rapidement avec le modèle, l'architecture et les composants de programmation qui composent le .NET Framework.

## Outils et débogueurs

Les [outils et débogueurs](#) du Kit de développement .NET Framework SDK vous permettent de créer, de déployer et de gérer des applications et des composants qui ciblent le .NET Framework.

## Commentaire

Nous travaillons dur afin que les technologies .NET Framework et le Kit de développement vous permettent de générer des applications puissantes. Vos commentaires nous sont extrêmement précieux. Envoyez-nous vos [commentaires et suggestions](#). Vous pouvez également visiter les [newsqroups](#) du Kit de développement .NET Framework SDK.

Pour obtenir les dernières informations sur les problèmes connus, consultez les [Notes de mise à jour](#).

Le lien *.NET Framework SDK Documentation* est celui qu'il faut suivre pour avoir une vue d'ensemble des classes .NET :

Page d'accueil du Kit de développement .NET Framework SDK

## Kit de développement .NET Framework SDK

Bienvenue dans Microsoft® .NET Framework version 1.1. .NET Framework est le modèle de programmation de la plateforme .NET. Les principaux composants de .NET Framework sont le Common Language Runtime et la bibliothèque de classes .NET Framework, qui contient ADO.NET, ASP.NET et Windows Forms. .NET Framework fournit un environnement d'exécution managé, un développement et un déploiement simplifiés et l'intégration à une grande variété de langages de programmation. Pour une introduction succincte à l'architecture de .NET Framework, consultez [Vue d'ensemble du .NET Framework](#).



Le Kit de développement de logiciel (SDK) .NET Framework contient des exemples, des compilateurs et des outils de ligne de commande conçus pour vous aider à créer des applications et des services basés sur la technologie .NET Framework. Le Kit de développement de logiciel (SDK) fournit également une documentation qui contient une référence complète sur les bibliothèques de classes, des vues d'ensemble conceptuelles, des procédures pas à pas, des informations sur les outils, ainsi que des didacticiels illustrant la création de types d'applications spécifiques. Pour trouver les informations qui vous intéressent, consultez la liste suivante des principales rubriques. Outre la liste suivante, consultez [Table de documentation par technologie](#) pour trouver des informations sur les principales fonctionnalités de .NET Framework.

Là, on suivra le lien *[Bibliothèque de classes]*. On y trouve la liste de toutes les classes de .NET :

Page d'accueil du Kit de développement .NET Framework SDK

## Kit de développement .NET Framework SDK

### Référence

Fournit une documentation de référence exhaustive sur le .NET Framework, en abordant notamment les sections principales suivantes :

- [Bibliothèque de classes](#)

Fournit la syntaxe, des exemples de code et des informations associées pour chaque classe contenue dans les espaces de noms .NET Framework.

Suivons par exemple le lien *System.Collections*. Cet espace de noms regroupe diverses classes implémentant des collections dont la classe *HashTable* :

Bibliothèque de classes .NET Framework

## Bibliothèque de classes

### [System.CodeDom.Compiler](#)

Contient des types permettant de gérer la génération et la compilation de code source dans les langages de programmation pris en charge. Chaque générateur de code peut produire du code source dans un langage de programmation particulier en fonction de la structure des modèles de code source CodeDOM (Code Document Object Model) se composant des éléments fournis par l'espace de noms [System.CodeDOM](#).

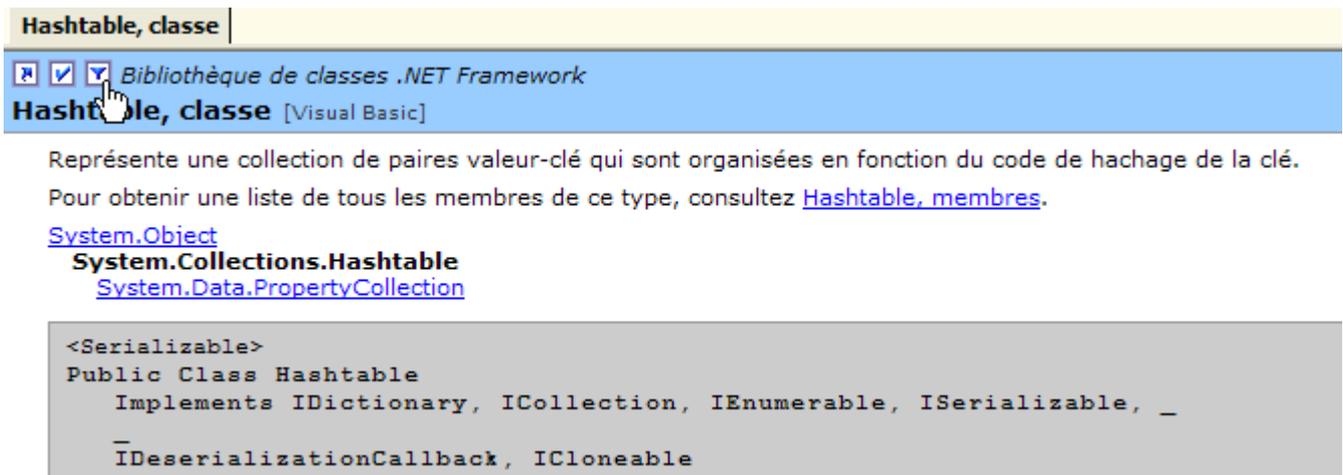
### [System.Collections](#)

Contient des interfaces et des classes qui définissent différentes collections d'objets, telles que des listes, des files d'attente, des tableaux de bits, des tables de hachage et des dictionnaires.

Suivons le lien *HashTable* ci-dessous :



Nous obtenons la page suivante :

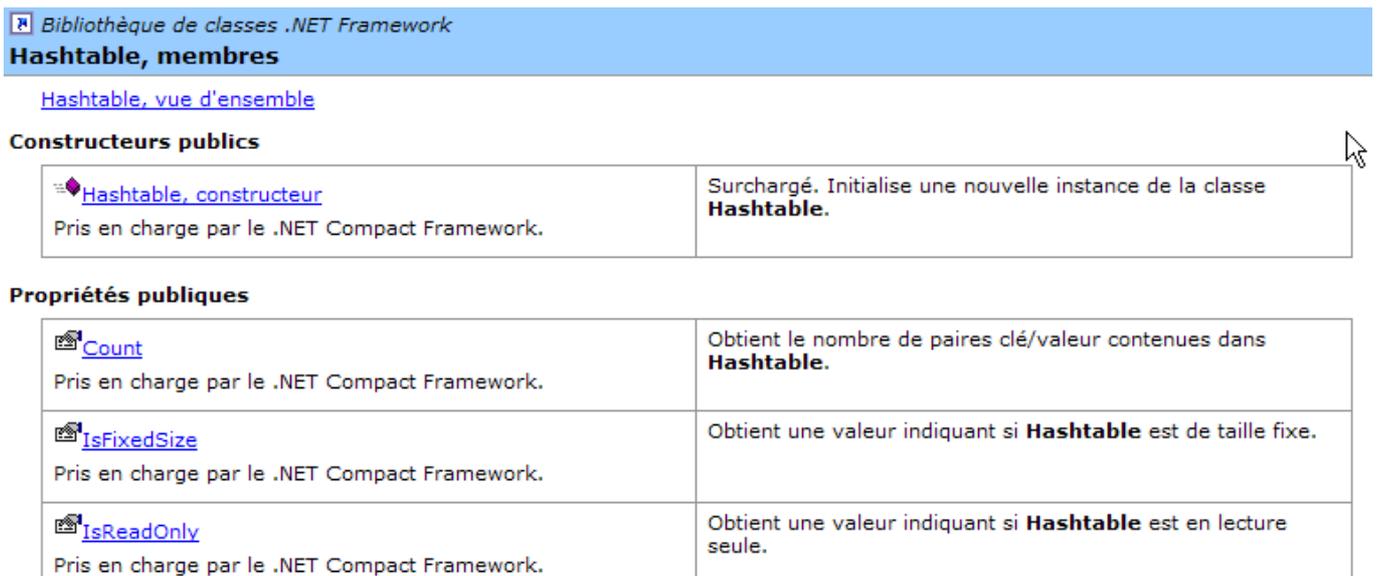


On remarquera ci-dessous, la position de la main. Elle pointe sur un lien permettant de préciser le langage désiré, ici Visual Basic. On y trouve le prototype de la classe ainsi que des exemples d'utilisation. Suivons le lien [Hashtable, membres] ci-dessous :

#### Voir aussi

[Hashtable, membres](#) | [System.Collections, espace de noms](#) | [IDictionary](#) | [IHashCodeProvider](#) | [Object.GetHashCode](#) | [Object.Equals](#) | [DictionaryEntry](#) | [Hashtable, membres \(syntaxe Visual J#\)](#) | [Programmation des extensions managées pour C++](#)

On obtient la description complète de la classe :



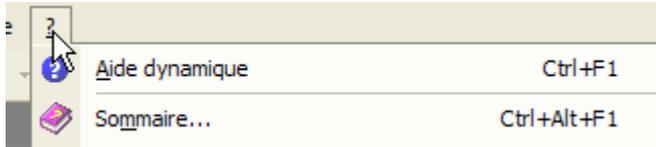
Cette méthode est la meilleure pour découvrir le SDK et ses classes. L'outil WinCV s'avère utile lorsqu'on connaît déjà un peu la classe et qu'on a oublié certains de ses membres. WinCV permet alors de retrouver rapidement la classe et ses membres.

## 3.2 Chercher de l'aide sur les classes avec VS.NET

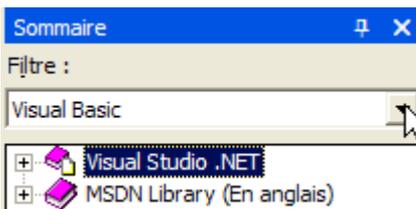
Nous donnons ici quelques indications pour trouver de l'aide avec Visual Studio.NET

### 3.2.1 Option Aide

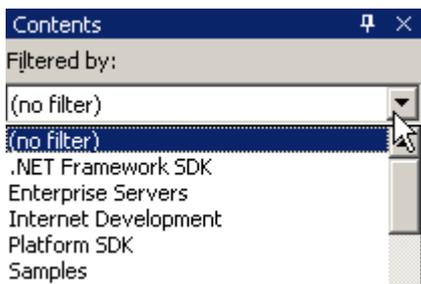
Prenons l'option [?] du menu.



On obtient la fenêtre suivante :



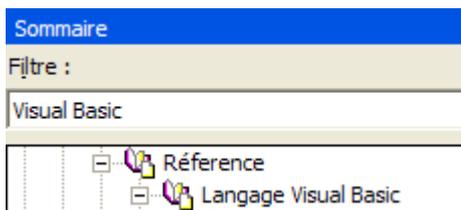
Dans la liste déroulante, on peut choisir un filtre d'aide. Ici, on prendra le filtre [Visual Basic].



Deux aides sont utiles :

- l'aide sur le langage VB.NET lui-même (syntaxe)
- l'aide sur les classes.NET utilisables par le langage VB.NET

L'aide au langage VB.NET est accessible via [Visual Studio.NET/Visual Basic et Visual C#/Reference/Visual Basic] :



On obtient la page d'aide suivante :

## Langage Visual Basic

Visual Basic .NET, la nouvelle génération du langage Visual Basic, représente un moyen simple et rapide de créer des applications .NET, y compris des services Web XML et des applications Web.

Visual Basic .NET comporte de nombreuses fonctionnalités nouvelles et améliorées, telles que l'héritage, les interfaces et la surcharge, qui en font un langage de programmation orienté objet puissant. Le modèle de threads libres et la gestion structurée des exceptions comptent aussi parmi les nouvelles fonctionnalités de langage. Visual Basic .NET intègre aussi entièrement le .NET Framework et le Common Language Runtime, qui offrent l'interopérabilité entre les langages, le garbage collection, une sécurité améliorée et une meilleure prise en charge du versioning.

### Dans cette section

A partir de là, les différentes sous-rubriques nous permettent d'avoir de l'aide sur différents thèmes de VB.NET. On prêter attention aux tutoriels de VB.NET :

### Rubriques connexes

[Introduction à Visual Studio .NET](#)

Offre des informations sur les nouvelles du .NET Framework ainsi que des conseils.

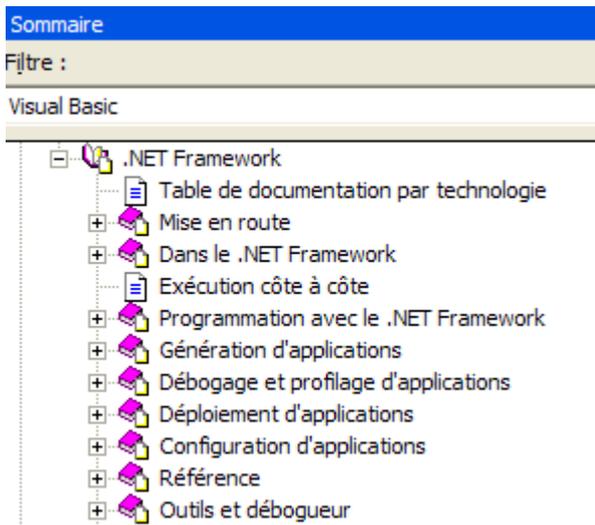
[Développement avec Visual Studio .NET](#)

Décrit les outils partagés qui permettent de développer.

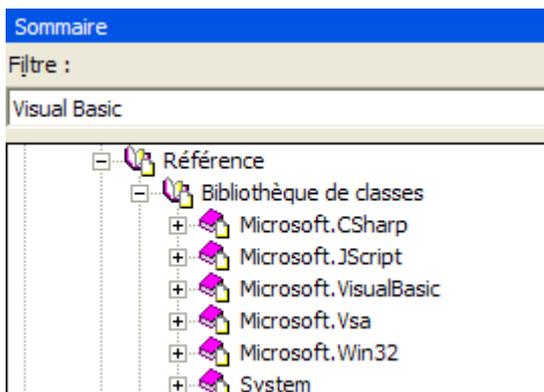
[Exemples et procédures pas à pas](#)

Montre comment créer des applications riches, des services Web et y accéder. Fournit également des exemples de Fitch and Mather.

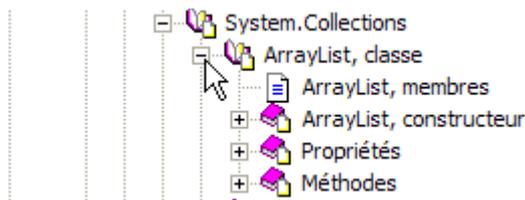
Pour avoir accès aux différentes classes de la plate-forme .NET, on choisira l'aide [Visual Studio.NET/.NET Framework].



On s'intéressera notamment à la rubrique [Référence/Bibliothèque de classes] :



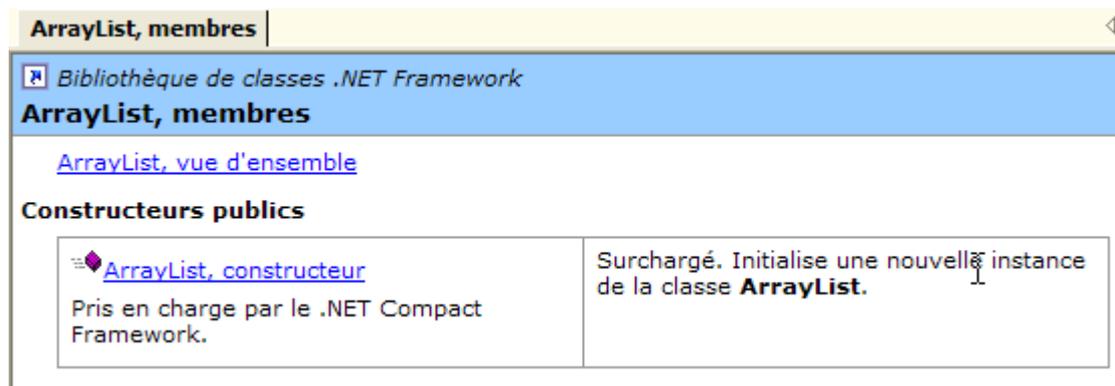
Supposons qu'on s'intéresse à la classe [ArrayList]. Elle se trouve dans l'espace de noms [System.Collections]. Il faut le savoir sinon on préférera la méthode de recherche exposée ci-après. On obtient l'aide suivante :



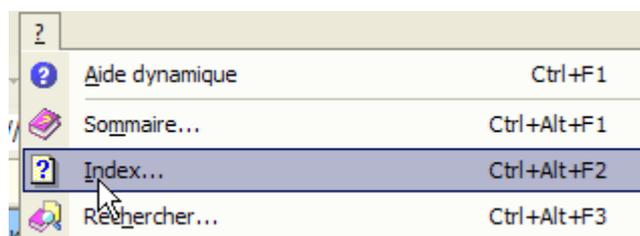
Le lien [ArrayList, classe] donne une vue d'ensemble de la classe :



Ce type de page existe pour toutes les classes. Elle donne un résumé de la classe avec des exemples. Pour une description des membres de la classe, on suivra le lien [ArrayList, membres] :



### 3.2.2 Aide/Index



L'option [Aide/index] permet de chercher une aide plus ciblée que l'aide précédente. Il suffit de taper le mot clé cherché :

Index

Rechercher :

ArrayList

Filtre :

Visual Basic

ArrayList (classe)

- à propos de la classe ArrayList
- constructeur
- exemples
- méthodes
- propriétés
- regrouper des données dans des collections
- tous les membres

L'avantage de cette méthode par rapport à la précédente est qu'on n'a pas besoin de savoir où se trouve ce qu'on cherche dans le système d'aide. C'est probablement la méthode à préférer lorsqu'on fait une recherche ciblée, l'autre méthode étant plus appropriée à une découverte de tout ce que propose l'aide.

### 3.3 La classe String

La classe **String** présente de nombreuses propriétés et méthodes. En voici quelques-unes :

Public ReadOnly Property Length As Integer	nombre de caractères de la chaîne
Public Default ReadOnly Property Chars(ByVal index As Integer) As Char	propriété indexée par défaut. [String].Chars(i) est le caractère n° i de la chaîne
Public Function EndsWith(ByVal value As String) As Boolean	rend vrai si la chaîne se termine par value
Public Function StartsWith(ByVal value As String) As Boolean	rend vrai si la chaîne commence par value
Overloads Public Function Equals(ByVal value As String) As Boolean	rend vrai si la chaîne est égale à value
Overloads Public Function IndexOf(ByVal value As String) As Integer	rend la première position dans la chaîne de la chaîne value - la recherche commence à partir du caractère n° 0
Overloads Public Function IndexOf(ByVal value As String, ByVal startIndex As Integer) As Integer	rend la première position dans la chaîne de la chaîne value - la recherche commence à partir du caractère n° startIndex
Overloads Public Shared Function Join(ByVal separator As String, ByVal value() As String) As String	méthode de classe - rend une chaîne de caractères, résultat de la concaténation des valeurs du tableau value avec le séparateur separator
Overloads Public Function Replace(ByVal oldChar As Char, ByVal newChar As Char) As String	rend une chaîne copie de la chaîne courante où le caractère oldChar a été remplacé par le caractère newChar
Overloads Public Function Split(ByVal ParamArray separator() As Char) As String()	la chaîne est vue comme une suite de champs séparés par les caractères présents dans le tableau separator. Le résultat est le tableau de ces champs
Overloads Public Function Substring(ByVal startIndex As Integer, ByVal length As Integer) As String	sous-chaîne de la chaîne courante commençant à la position startIndex et ayant length caractères
Overloads Public Function ToLower() As String	rend la chaîne courante en minuscules
Overloads Public Function ToUpper() As String	rend la chaîne courante en majuscules
Overloads Public Function Trim() As String	rend la chaîne courante débarrassée de ses espaces de début et fin

Une chaîne C peut être considérée comme un tableau de caractères. Ainsi

- **C.Chars(i)** est le caractère i de C
- **C.Length** est le nombre de caractères de C

Considérons l'exemple suivant :

Exemples de classes .NET

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
Sub Main()
    Dim uneChaine As String = "l'oiseau vole au-dessus des nuages"
    affiche("uneChaine=" + uneChaine)
    affiche("uneChaine.Length=" & uneChaine.Length)
    affiche("chaine[10]=" + uneChaine.Chars(10))
    affiche("uneChaine.IndexOf("vole")=" & uneChaine.IndexOf("vole"))
    affiche("uneChaine.IndexOf("x")=" & uneChaine.IndexOf("x"))
    affiche("uneChaine.LastIndexOf('a')=" & uneChaine.LastIndexOf("a"c))
    affiche("uneChaine.LastIndexOf('x')=" & uneChaine.LastIndexOf("x"c))
    affiche("uneChaine.Substring(4,7)=" + uneChaine.Substring(4, 7))
    affiche("uneChaine.ToUpper()=" + uneChaine.ToUpper())
    affiche("uneChaine.ToLower()=" + uneChaine.ToLower())
    affiche("uneChaine.Replace('a','A')=" + uneChaine.Replace("a"c, "A"c))
    Dim champs As String() = uneChaine.Split(Nothing)
    Dim i As Integer
    For i = 0 To champs.Length - 1
        affiche("champs[" & i & "]=[" & champs(i) & "]")
    Next i
    affiche("Join("":", champs)=" + System.String.Join(":", champs))
    affiche("(" abc ").Trim()=[" + " abc ".Trim() + "]")
End Sub

' affiche
Sub affiche(ByVal msg As [String])
    ' affiche msg
    Console.Out.WriteLine(msg)
End Sub
End Module

```

L'exécution donne les résultats suivants :

```

dos>vbc string1.vb

dos>string1
uneChaine=l'oiseau vole au-dessus des nuages
uneChaine.Length=34
chaine[10]=o
uneChaine.IndexOf("vole")=9
uneChaine.IndexOf("x")=-1
uneChaine.LastIndexOf('a')=30
uneChaine.LastIndexOf('x')=-1
uneChaine.Substring(4,7)=seau vo
uneChaine.ToUpper()=L'OISEAU VOLE AU-DESSUS DES NUAGES
uneChaine.ToLower()=l'oiseau vole au-dessus des nuages
uneChaine.Replace('a','A')=l'oiseAu vole Au-dessus des nuAges
champs[0]=[l'oiseau]
champs[1]=[vole]
champs[2]=[au-dessus]
champs[3]=[des]
champs[4]=[nuages]
Join(":", champs)=l'oiseau:vole:au-dessus:des:nuages
(" abc ").Trim()=[abc]

```

Considérons un nouvel exemple :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module string2
Sub Main()
    ' la ligne à analyser
    Dim ligne As String = "un:deux::trois:"
    ' les séparateurs de champs
    Dim séparateurs() As Char = {"":c}
    ' split
    Dim champs As String() = ligne.Split(séparateurs)

```

```

Dim i As Integer
For i = 0 To champs.Length - 1
    Console.Out.WriteLine(("Champs[" & i & "]=" & champs(i))
Next i
' join
Console.Out.WriteLine(("join=[" + System.String.Join(":", champs) + "]"))
End Sub
End Module

```

et les résultats d'exécution :

```

Champs[0]=un
Champs[1]=deux
Champs[2]=
Champs[3]=trois
Champs[4]=
join=[un:deux::trois:]

```

La méthode **Split** de la classe *String* permet de mettre dans un tableau les champs d'une chaîne de caractères. La définition de la méthode utilisée ici est la suivante :

```
Overloads Public Function Split(ByVal ParamArray separator() As Char) As String()
```

**separator** | tableau de caractères. Ces caractères représentent les caractères utilisés pour séparer les champs de la chaîne de caractères. Ainsi si la chaîne est [champ1, champ2, champ3] on pourra utiliser *separator=new char() {"", "c"}*. Si le séparateur est une suite d'espaces on utilisera *separator=nothing*.

**résultat** | tableau de chaînes de caractères où chaque élément est un champ de la chaîne.

La méthode **Join** est une méthode statique de la classe *String* :

```
Overloads Public Shared Function Join(ByVal separator As String,ByVal value() As String) As String
```

**value** | tableau de chaînes de caractères

**separator** | une chaîne de caractères qui servira de séparateur de champs

**résultat** | une chaîne de caractères formée de la concaténation des éléments du tableau *value* séparés par la chaîne *separator*.

## 3.4 La classe Array

La classe **Array** implémente un tableau. Nous utiliserons dans notre exemple les propriétés et méthodes suivantes :

```
Public ReadOnly Property Length As Integer
```

propriété - nombre d'éléments du tableau

```
Overloads Public Shared Function BinarySearch(ByVal array As Array,ByVal index As Integer,ByVal length As Integer,ByVal value As Object) As Integer
```

méthode de classe - rend la position de *value* dans le tableau trié *array* - cherche à partir de la position *index* et avec *length* éléments

```
Overloads Public Shared Sub Copy(ByVal sourceArray As Array,ByVal destinationArray As Array,ByVal length As Integer)
```

méthode de classe - copie *length* éléments de *sourceArray* dans *destinationArray* - *destinationArray* est créé pour l'occasion

```
Overloads Public Shared Sub Sort(ByVal array As Array)
```

méthode de classe - trie le tableau *array* - celui doit contenir un type de données ayant une fonction d'ordre par défaut (chaînes, nombres).

Le programme suivant illustre l'utilisation de la classe *Array* :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Module test
    Sub Main()
        ' lecture des éléments d'un tableau tapés au clavier
        Dim terminé As [Boolean] = False
        Dim i As Integer = 0

```

```

Dim éléments1 As Double() = Nothing
Dim éléments2 As Double() = Nothing
Dim élément As Double = 0
Dim réponse As String = Nothing
Dim erreur As [Boolean] = False

While Not terminé
    ' question
    Console.Out.WriteLine("Élément (réel) " & i & " du tableau (rien pour terminer) : ")
    ' lecture de la réponse
    réponse = Console.ReadLine().Trim()
    ' fin de saisie si chaîne vide
    If réponse.Equals("") Then
        Exit While
    End If
    ' vérification saisie
    Try
        élément = [Double].Parse(réponse)
        erreur = False
    Catch
        Console.Error.WriteLine("Saisie incorrecte, recommencez")
        erreur = True
    End Try
    ' si pas d'erreur
    If Not erreur Then
        ' nouveau tableau pour accueillir le nouvel élément
        éléments2 = New Double(i) {}
        ' copie ancien tableau vers nouveau tableau
        If i <> 0 Then
            Array.Copy(éléments1, éléments2, i)
        End If
        ' nouveau tableau devient ancien tableau
        éléments1 = éléments2
        ' plus besoin du nouveau tableau
        éléments2 = Nothing
        ' insertion nouvel élément
        éléments1(i) = élément
        ' un élémnt de plus dans le tableau
        i += 1
    End If
End While
' affichage tableau non trié
System.Console.Out.WriteLine("Tableau non trié")
For i = 0 To éléments1.Length - 1
    Console.Out.WriteLine(("éléments[" & i & "]=" & éléments1(i)))
Next i
' tri du tableau
System.Array.Sort(éléments1)
' affichage tableau trié
System.Console.Out.WriteLine("Tableau trié")
For i = 0 To éléments1.Length - 1
    Console.Out.WriteLine(("éléments[" & i & "]=" & éléments1(i)))
Next i
' Recherche
While Not terminé
    ' question
    Console.Out.WriteLine("Élément cherché (rien pour arrêter) : ")
    ' lecture-vérification réponse
    réponse = Console.ReadLine().Trim()
    ' fini ?
    If réponse.Equals("") Then
        Exit While
    End If
    ' vérification
    Try
        élément = [Double].Parse(réponse)
        erreur = False
    Catch
        Console.Error.WriteLine("Saisie incorrecte, recommencez")
        erreur = True
    End Try
    ' si pas d'erreur
    If Not erreur Then
        ' on cherche l'élément dans le tableau trié
        i = System.Array.BinarySearch(éléments1, 0, éléments1.Length, élément)
        ' Affichage réponse
        If i >= 0 Then
            Console.Out.WriteLine(("Trouvé en position " & i))
        Else
            Console.Out.WriteLine("Pas dans le tableau")
        End If
    End If
End While

```

```

End If
End While
End Sub
End Module

```

Les résultats écran sont les suivants :

```

Élément (réel) 0 du tableau (rien pour terminer) : 10,4
Élément (réel) 1 du tableau (rien pour terminer) : 5,2
Élément (réel) 2 du tableau (rien pour terminer) : 8,7
Élément (réel) 3 du tableau (rien pour terminer) : 3,6
Élément (réel) 4 du tableau (rien pour terminer) :
Tableau non trié
éléments[0]=10,4
éléments[1]=5,2
éléments[2]=8,7
éléments[3]=3,6
Tableau trié
éléments[0]=3,6
éléments[1]=5,2
éléments[2]=8,7
éléments[3]=10,4
Élément cherché (rien pour arrêter) : 8,7
Trouvé en position 2
Élément cherché (rien pour arrêter) : 11
Pas dans le tableau
Élément cherché (rien pour arrêter) : a
Saisie incorrecte, recommencez
Élément cherché (rien pour arrêter) :

```

La première partie du programme construit un tableau à partir de données numériques tapées au clavier. Le tableau ne peut être dimensionné à priori puisqu'on ne connaît pas le nombre d'éléments que va taper l'utilisateur. On travaille alors avec deux tableaux *éléments1* et *éléments2*.

```

' nouveau tableau pour accueillir le nouvel élément
éléments2 = New Double(i) {}
' copie ancien tableau vers nouveau tableau
If i <> 0 Then
    Array.Copy(éléments1, éléments2, i)
End If
' nouveau tableau devient ancien tableau
éléments1 = éléments2
' plus besoin du nouveau tableau
éléments2 = Nothing
' insertion nouvel élément
éléments1(i) = élément
' un élémt de plus dans le tableau
i += 1

```

Le tableau *éléments1* contient les valeurs actuellement saisies. Lorsque l'utilisateur ajoute une nouvelle valeur, on construit un tableau *éléments2* avec un élément de plus que *éléments1*, on copie le contenu de *éléments1* dans *éléments2* (`Array.Copy`), on fait "pointer" *éléments1* sur *éléments2* et enfin on ajoute le nouvel élément à *éléments1*. C'est un processus complexe qui peut être simplifié si au lieu d'utiliser un tableau de taille fixe (*Array*) on utilise un tableau de taille variable (*ArrayList*).

Le tableau est trié avec la méthode **Array.Sort**. Cette méthode peut être appelée avec différents paramètres précisant les règles de tri. Sans paramètres, c'est ici un tri en ordre croissant qui est fait par défaut. Enfin, la méthode **Array.BinarySearch** permet de chercher un élément dans un tableau trié.

## 3.5 La classe ArrayList

La classe *ArrayList* permet d'implémenter des tableaux de taille variable au cours de l'exécution du programme, ce que ne permet pas la classe *Array* précédente. Voici quelques-unes des propriétés et méthodes courantes :

Public Sub New()	construit une liste vide
Public Overridable ReadOnly Property Count As Integer Implements ICollection.Count	nombre d'éléments de la collection
Public Overridable Function Add(ByVal value As Object) As Integer Implements IList.Add	ajoute l'objet value à la fin de la collection
Public Overridable Sub Clear() Implements IList.Clear	efface la liste

Overloads Public Overridable Function IndexOf(ByVal value As Object) As Integer Implements IList.IndexOf	indice de l'objet value dans la liste ou -1 s'il n'existe pas
Overloads Public Overridable Function IndexOf(ByVal value As Object, ByVal startIndex As Integer) As Integer	idem mais en cherchant à partir de l'élément n° startIndex
Overloads Public Overridable Function LastIndexOf(ByVal value As Object) As Integer	idem mais rend l'indice de la dernière occurrence de value dans la liste
Overloads Public Overridable Function LastIndexOf(ByVal value As Object, ByVal startIndex As Integer) As Integer	idem mais en cherchant à partir de l'élément n° startIndex
Public Overridable Sub Remove( ByVal obj As Object) Implements IList.Remove	enlève l'objet obj s'il existe dans la liste
Public Overridable Sub RemoveAt(ByVal index As Integer) Implements IList.RemoveAt	enlève l'élément index de la liste
Overloads Public Overridable Function BinarySearch(ByVal value As Object) As Integer	rend la position de l'objet value dans la liste ou -1 s'il ne s'y trouve pas. La liste doit être triée
Overloads Public Overridable Sub Sort()	trie la liste. Celle-ci doit contenir des objets ayant une relation d'ordre prédéfinie (chaînes, nombres)
Overloads Public Overridable Sub Sort(ByVal comparer As IComparer)	trie la liste selon la relation d'ordre établie par la fonction comparer

Reprenons l'exemple traité avec des objets de type *Array* et traitons-le avec des objets de type *ArrayList* :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections

Module test
Sub Main()
' lecture des éléments d'un tableau tapés au clavier
Dim terminé As [Boolean] = False
Dim i As Integer = 0
Dim éléments As New ArrayList
Dim élément As Double = 0
Dim réponse As String = Nothing
Dim erreur As [Boolean] = False

While Not terminé
' question
Console.Out.Write(("Élément (réel) " & i & " du tableau (rien pour terminer) : "))
' lecture de la réponse
réponse = Console.ReadLine().Trim()
' fin de saisie si chaîne vide
If réponse.Equals("") Then
Exit While
End If
' vérification saisie
Try
élément = Double.Parse(réponse)
erreur = False
Catch
Console.Error.WriteLine("Saisie incorrecte, recommencez")
erreur = True
End Try
' si pas d'erreur
If Not erreur Then
' un élém de plus dans le tableau
éléments.Add(élément)
End If
End While
' affichage tableau non trié
```

```

System.Console.Out.WriteLine("Tableau non trié")
For i = 0 To éléments.Count - 1
    Console.Out.WriteLine(("éléments[" & i & "]=" & éléments(i).ToString))
Next i ' tri du tableau
éléments.Sort()
' affichage tableau trié
System.Console.Out.WriteLine("Tableau trié")
For i = 0 To éléments.Count - 1
    Console.Out.WriteLine(("éléments[" & i & "]=" & éléments(i).ToString))
Next i
' Recherche
While Not terminé
    ' question
    Console.Out.Write("Élément cherché (rien pour arrêter) : ")
    ' lecture-vérification réponse
    réponse = Console.ReadLine().Trim()
    ' fini ?
    If réponse.Equals("") Then
        Exit While
    End If
    ' vérification
    Try
        élément = [Double].Parse(réponse)
        erreur = False
    Catch
        Console.Error.WriteLine("Saisie incorrecte, recommencez")
        erreur = True
    End Try
    ' si pas d'erreur
    If Not erreur Then
        ' on cherche l'élément dans le tableau trié
        i = éléments.BinarySearch(élément)
        ' Affichage réponse
        If i >= 0 Then
            Console.Out.WriteLine(("Trouvé en position " & i))
        Else
            Console.Out.WriteLine("Pas dans le tableau")
        End If
    End If
End While
End Sub
End Module

```

Les résultats d'exécution sont les suivants :

```

Élément (réel) 0 du tableau (rien pour terminer) : 10,4
Élément (réel) 0 du tableau (rien pour terminer) : 5,2
Élément (réel) 0 du tableau (rien pour terminer) : a
Saisie incorrecte, recommencez
Élément (réel) 0 du tableau (rien pour terminer) : 3,7
Élément (réel) 0 du tableau (rien pour terminer) : 15
Élément (réel) 0 du tableau (rien pour terminer) :
Tableau non trié
éléments[0]=10,4
éléments[1]=5,2
éléments[2]=3,7
éléments[3]=15
Tableau trié
éléments[0]=3,7
éléments[1]=5,2
éléments[2]=10,4
éléments[3]=15
Élément cherché (rien pour arrêter) : a
Saisie incorrecte, recommencez
Élément cherché (rien pour arrêter) : 15
Trouvé en position 3
Élément cherché (rien pour arrêter) : 1
Pas dans le tableau
Élément cherché (rien pour arrêter) :

```

## 3.6 La classe Hashtable

La classe *Hashtable* permet d'implémenter un dictionnaire. On peut voir un dictionnaire comme un tableau à deux colonnes :

clé	valeur
clé1	valeur1
clé2	valeur2
..	...

Les clés sont uniques, c.a.d. qu'il ne peut y avoir deux clés identiques. Les méthodes et propriétés principales de la classe **Hashtable** sont les suivantes :

<code>Public Sub New()</code>	crée un dictionnaire vide
<code>Public Overridable Sub Add(ByVal key As Object,ByVal value As Object) Implements IDictionary.Add</code>	ajoute une ligne (key,value) dans le dictionnaire où key et value sont des références d'objets.
<code>Public Overridable Sub Remove(ByVal key As Object) Implements IDictionary.Remove</code>	élimine du dictionnaire la ligne de clé=key
<code>Public Overridable Sub Clear() Implements IDictionary.Clear</code>	vide le dictionnaire
<code>Public Overridable Function ContainsKey(ByVal key As Object) As Boolean</code>	rend vrai (true) si la clé key appartient au dictionnaire.
<code>Public Overridable Function ContainsValue(ByVal value As Object) As Boolean</code>	rend vrai (true) si la valeur value appartient au dictionnaire.
<code>Public Overridable ReadOnly Property Count As Integer Implements ICollection.Count</code>	propriété : nombre d'éléments du dictionnaire (clé,valeur)
<code>Public Overridable ReadOnly Property Keys As ICollection Implements IDictionary.Keys</code>	propriété : collection des clés du dictionnaire
<code>Public Overridable ReadOnly Property Values As ICollection Implements IDictionary.Values</code>	propriété : collection des valeurs du dictionnaire
<code>Public Overridable Default Property Item(ByVal key As Object) As Object Implements IDictionary.Item</code>	propriété indexée : permet de connaître ou de fixer la valeur associée à une clé key

Considérons le programme exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections

Module test
Sub Main()
    Dim liste() As [String] = {"jean:20", "paul:18", "mélanie:10", "violette:15"}
    Dim i As Integer
    Dim champs As [String]() = Nothing
    Dim séparateurs() As Char = {":"c}
    ' remplissage du dictionnaire
    Dim dico As New Hashtable
    For i = 0 To liste.Length - 1
        champs = liste(i).Split(séparateurs)
        dico.Add(champs(0), champs(1))
    Next i
    ' nbre d'éléments dans le dictionnaire
    Console.Out.WriteLine("Le dictionnaire a " & dico.Count & " éléments")
    ' liste des clés
    Console.Out.WriteLine("[Liste des clés]")
    Dim clés As IEnumerator = dico.Keys.GetEnumerator()
    While clés.MoveNext()
        Console.Out.WriteLine(clés.Current)
    End While
    ' liste des valeurs
    Console.Out.WriteLine("[Liste des valeurs]")
```

```

Dim valeurs As IEnumerator = dico.Values.GetEnumerator()
While valeurs.MoveNext()
    Console.Out.WriteLine(valeurs.Current)
End While
' liste des clés & valeurs
Console.Out.WriteLine("[Liste des clés & valeurs]")
clés.Reset()
While clés.MoveNext()
    Console.Out.WriteLine(("clé=" & clés.Current.ToString & " valeur=" & dico(clés.Current).ToString))
End While
' on supprime la clé "paul"
Console.Out.WriteLine("[Suppression d'une clé]")
dico.Remove("paul")
' liste des clés & valeurs
Console.Out.WriteLine("[Liste des clés & valeurs]")
clés = dico.Keys.GetEnumerator()
While clés.MoveNext()
    Console.Out.WriteLine(("clé=" & clés.Current.ToString & " valeur=" & dico(clés.Current).ToString))
End While

' recherche dans le dictionnaire
Dim nomCherché As [String] = Nothing
Console.Out.Write("Nom recherché (rien pour arrêter) : ")
nomCherché = Console.ReadLine().Trim()
Dim value As [Object] = Nothing
While Not nomCherché.Equals("")
    If dico.ContainsKey(nomCherché) Then
        value = dico(nomCherché)
        Console.Out.WriteLine((nomCherché + "," + CType(value, [String])))
    Else
        Console.Out.WriteLine(("Nom " + nomCherché + " inconnu"))
    End If
    ' recherche suivante
    Console.Out.Write("Nom recherché (rien pour arrêter) : ")
    nomCherché = Console.ReadLine().Trim()
End While
End Sub
End Module

```

Les résultats d'exécution sont les suivants :

```

Le dictionnaire a 4 éléments
[Liste des clés]
mélanie
paul
violette
jean
[Liste des valeurs]
10
18
15
20
[Liste des clés & valeurs]
clé=mélanie valeur=10
clé=paul valeur=18
clé=violette valeur=15
clé=jean valeur=20
[Suppression d'une clé]
[Liste des clés & valeurs]
clé=mélanie valeur=10
clé=violette valeur=15
clé=jean valeur=20
Nom recherché (rien pour arrêter) : paul
Nom paul inconnu
Nom recherché (rien pour arrêter) : mélanie
mélanie,10
Nom recherché (rien pour arrêter) :

```

Le programme utilise également un objet **Ienumerator** pour parcourir les collections de clés et de valeurs du dictionnaire de type **ICollection** (cf ci-dessus les propriétés Keys et Values). Une collection est un ensemble d'objets qu'on peut parcourir. L'interface *ICollection* est définie comme suit :

**ICollection, membres**[ICollection, vue d'ensemble](#)**Propriétés publiques**

 <a href="#">Count</a> Pris en charge par le .NET Compact Framework.	En cas d'implémentation par une classe, obtient le nombre d'éléments contenus dans <b>ICollection</b> .
--	---

La propriété *Count* nous permet de connaître le nombre d'éléments de la collection. L'interface *ICollection* dérive de l'interface *IEnumerable* :

**IEnumerable, membres**[IEnumerable, vue d'ensemble](#)**Méthodes publiques**

 <a href="#">GetEnumerator</a> Pris en charge par le .NET Compact Framework.	Retourne un énumérateur qui peut itérer sur une collection.
--	---

Cette interface n'a qu'une méthode **GetEnumerator** qui nous permet d'obtenir un objet de type **IEnumerator** :

**IEnumerator, membres**[IEnumerator, vue d'ensemble](#)**Propriétés publiques**

 <a href="#">Current</a> Pris en charge par le .NET Compact Framework.	Obtient l'élément en cours dans la collection.
--	--

**Méthodes publiques**

 <a href="#">MoveNext</a> Pris en charge par le .NET Compact Framework.	Avance l'énumérateur à l'élément suivant de la collection.
 <a href="#">Reset</a> Pris en charge par le .NET Compact Framework.	Rétablit l'énumérateur à sa position initiale, qui précède le premier élément de la collection.

La méthode *GetEnumerator()* d'une collection *ICollection* nous permet de parcourir la collection avec les méthodes suivantes :

MoveNext	positionne sur l'élément suivant de la collection. Rend vrai (true) si cet élément existe, faux (false) sinon. Le premier <i>MoveNext</i> positionne sur le 1er élément. L'élément "courant" de la collection est alors disponible dans la propriété <i>Current</i> de l'énumérateur
Current	propriété : élément courant de la collection
Reset	repositionne l'énumérateur en début de collection, c.a.d. avant le 1er élément.

La structure d'itération sur les éléments d'une collection (*ICollection*) *C* est donc la suivante :

```
' définir la collection
dim C as ICollection C=...
' obtenir un énumérateur de cette collection
```

```

dim itérateur as IEnumerator=C.GetEnumerator();
' parcourir la collection avec cet énumérateur
while(itérateur.MoveNext())
' on a un élément courant
' exploiter itérateur.Current
end while

```

## 3.7 La classe StreamReader

La classe *StreamReader* permet de lire le contenu d'un fichier. Voici quelques-unes de ses propriétés et méthodes :

Public Sub New(ByVal path As String)	ouvre un flux à partir du fichier path. Une exception est lancée si celui-ci n'existe pas
Overrides Public Sub Close()	ferme le flux
Overrides Public Function ReadLine() As String	lit une ligne du flux ouvert
Overrides Public Function ReadToEnd() As String	lit le reste du flux depuis la position courante

Voici un exemple :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.IO

Module test
Sub Main()
Dim ligne As String = Nothing
Dim fluxInfos As StreamReader = Nothing
' lecture contenu du fichier
Try
fluxInfos = New StreamReader("infos.txt")
ligne = fluxInfos.ReadLine()
While Not (ligne Is Nothing)
System.Console.Out.WriteLine(ligne)
ligne = fluxInfos.ReadLine()
End While
Catch e As Exception
System.Console.Error.WriteLine("L'erreur suivante s'est produite : " & e.ToString)
Finally
Try
fluxInfos.Close()
Catch
End Try
End Try
End Sub
End Module

```

et ses résultats d'exécution :

```

dos>more infos.txt
12620:0:0
13190:0,05:631
15640:0,1:1290,5
24740:0,15:2072,5
31810:0,2:3309,5
39970:0,25:4900
48360:0,3:6898,5
55790:0,35:9316,5
92970:0,4:12106
127860:0,45:16754,5
151250:0,5:23147,5
172040:0,55:30710
195000:0,6:39312
0:0,65:49062

dos>file1
12620:0:0
13190:0,05:631

```

```

15640:0,1:1290,5
24740:0,15:2072,5
31810:0,2:3309,5
39970:0,25:4900
48360:0,3:6898,5
55790:0,35:9316,5
92970:0,4:12106
127860:0,45:16754,5
151250:0,5:23147,5
172040:0,55:30710
195000:0,6:39312
0:0,65:49062

```

### 3.8 La classe StreamWriter

La classe *StreamWriter* permet d'écrire dans fichier. Voici quelques-unes de ses propriétés et méthodes :

<code>Public Sub New(ByVal path As String)</code>	ouvre un flux d'écriture à partir du fichier path. Une exception est lancée si celui-ci ne peut être créé
<code>Public Overridable Property AutoFlush As Boolean</code>	si égal à vrai, l'écriture dans le flux ne passe pas par l'intermédiaire d'une mémoire tampon sinon l'écriture dans le flux n'est pas immédiate : il y a d'abord écriture dans une mémoire tampon puis dans le flux lorsque la mémoire tampon est pleine. Par défaut c'est le mode bufferisé qui est utilisé. Il convient bien pour les flux fichier mais généralement pas pour les flux réseau.
<code>Public Overridable Property NewLine As String</code>	pour fixer ou connaître la marque de fin de ligne à utiliser par la méthode WriteLine
<code>Overrides Public Sub Close()</code>	ferme le flux
<code>Overloads Public Overridable Sub WriteLine(ByVal value As String)</code>	écrit une ligne dans le flux d'écriture
<code>Overrides Public Sub Flush()</code>	écrit la mémoire tampon dans le flux

Considérons l'exemple suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.IO

Module test
Sub Main()
Dim ligne As String = Nothing ' une ligne de texte
Dim fluxInfos As StreamWriter = Nothing ' le fichier texte
Try
' création du fichier texte
fluxInfos = New StreamWriter("infos.txt")
' lecture ligne tapée au clavier
Console.Out.Write("ligne (rien pour arrêter) : ")
ligne = Console.In.ReadLine().Trim()
' boucle tant que la ligne saisie est non vide
While ligne <> ""
' écriture ligne dans fichier texte
fluxInfos.WriteLine(ligne)
' lecture nouvelle ligne au clavier
Console.Out.Write("ligne (rien pour arrêter) : ")
ligne = Console.In.ReadLine().Trim()
End While
Catch e As Exception
System.Console.Error.WriteLine("L'erreur suivante s'est produite : " & e.ToString)
Finally
' fermeture fichier
Try
fluxInfos.Close()
Catch

```

```
End Try
End Try
End Sub
End Module
```

et les résultats d'exécution :

```
dos>file2
ligne (rien pour arrêter) : ligne1
ligne (rien pour arrêter) : ligne2
ligne (rien pour arrêter) : ligne3
ligne (rien pour arrêter) :

dos>more infos.txt
ligne1
ligne2
ligne3
```

## 3.9 La classe Regex

La classe *Regex* permet l'utilisation d'expression régulières. Celles-ci permettent de tester le format d'une chaîne de caractères. Ainsi on peut vérifier qu'une chaîne représentant une date est bien au format *jj/mm/aa*. On utilise pour cela un modèle et on compare la chaîne à ce modèle. Ainsi dans cet exemple, *j* *m* et *a* doivent être des chiffres. Le modèle d'un format de date valide est alors `"\d\d/\d\d/\d\d"` où le symbole `\d` désigne un chiffre. Les symboles utilisables dans un modèle sont les suivants (documentation Microsoft) :

Caractère	Description
<code>\</code>	Marque le caractère suivant comme caractère spécial ou littéral. Par exemple, "n" correspond au caractère "n". "\n" correspond à un caractère de nouvelle ligne. La séquence "\\" correspond à "\", tandis que "\(" correspond à "(".
<code>^</code>	Correspond au début de la saisie.
<code>\$</code>	Correspond à la fin de la saisie.
<code>*</code>	Correspond au caractère précédent zéro fois ou plusieurs fois. Ainsi, "zo*" correspond à "z" ou à "zoo".
<code>+</code>	Correspond au caractère précédent une ou plusieurs fois. Ainsi, "zo+" correspond à "zoo", mais pas à "z".
<code>?</code>	Correspond au caractère précédent zéro ou une fois. Par exemple, "a?ve?" correspond à "ve" dans "lever".
<code>.</code>	Correspond à tout caractère unique, sauf le caractère de nouvelle ligne.
<code>(modèle)</code>	Recherche le <i>modèle</i> et mémorise la correspondance. La sous-chaîne correspondante peut être extraite de la collection <b>Matches</b> obtenue, à l'aide d'Item <b>[0]...[n]</b> . Pour trouver des correspondances avec des caractères entre parenthèses (), utilisez "\" ou "\".
<code>x y</code>	Correspond soit à <i>x</i> soit à <i>y</i> . Par exemple, "z foot" correspond à "z" ou à "foot". "(z f)oo" correspond à "zoo" ou à "foo".
<code>{n}</code>	<i>n</i> est un nombre entier non négatif. Correspond exactement à <i>n</i> fois le caractère. Par exemple, "o{2}" ne correspond pas à "o" dans "Bob," mais aux deux premiers "o" dans "fooooot".
<code>{n,}</code>	<i>n</i> est un entier non négatif. Correspond à au moins <i>n</i> fois le caractère. Par exemple, "o{2,}" ne correspond pas à "o" dans "Bob", mais à tous les "o" dans "fooooot". "o{1,}" équivaut à "o+" et "o{0,}" équivaut à "o*".
<code>{n,m}</code>	<i>m</i> et <i>n</i> sont des entiers non négatifs. Correspond à au moins <i>n</i> et à au plus <i>m</i> fois le caractère. Par exemple, "o{1,3}" correspond aux trois premiers "o" dans "fooooot" et "o{0,1}" équivaut à "o?".
<code>[xyz]</code>	Jeu de caractères. Correspond à l'un des caractères indiqués. Par exemple, "[abc]" correspond à "a" dans "plat".
<code>[^xyz]</code>	Jeu de caractères négatif. Correspond à tout caractère non indiqué. Par exemple, "[^abc]" correspond à "p" dans "plat".
<code>[a-z]</code>	Plage de caractères. Correspond à tout caractère dans la série spécifiée. Par exemple, "[a-z]" correspond à tout caractère alphabétique minuscule compris entre "a" et "z".
<code>[^m-z]</code>	Plage de caractères négative. Correspond à tout caractère ne se trouvant pas dans la série spécifiée. Par exemple, "[^m-z]" correspond à tout caractère ne se trouvant pas entre "m" et "z".
<code>\b</code>	Correspond à une limite représentant un mot, autrement dit, à la position entre un mot et un espace. Par exemple, "er\b" correspond à "er" dans "lever", mais pas à "er" dans "verbe".
<code>\B</code>	Correspond à une limite ne représentant pas un mot. "en*t\B" correspond à "ent" dans "bien entendu".
<code>\d</code>	Correspond à un caractère représentant un chiffre. Équivaut à [0-9].
<code>\D</code>	Correspond à un caractère ne représentant pas un chiffre. Équivaut à [^0-9].
<code>\f</code>	Correspond à un caractère de saut de page.
<code>\n</code>	Correspond à un caractère de nouvelle ligne.
<code>\r</code>	Correspond à un caractère de retour chariot.
<code>\s</code>	Correspond à tout espace blanc, y compris l'espace, la tabulation, le saut de page, etc. Équivaut à "[\f\n\r\t\v]".
<code>\S</code>	Correspond à tout caractère d'espace non blanc. Équivaut à "[^f\n\r\t\v]".
<code>\t</code>	Correspond à un caractère de tabulation.
<code>\v</code>	Correspond à un caractère de tabulation verticale.
<code>\w</code>	Correspond à tout caractère représentant un mot et incluant un trait de soulignement. Équivaut à "[A-Za-z0-9_]".
<code>\W</code>	Correspond à tout caractère ne représentant pas un mot. Équivaut à "[^A-Za-z0-9_]".
<code>\num</code>	Correspond à <i>num</i> , où <i>num</i> est un entier positif. Fait référence aux correspondances mémorisées. Par exemple, "( )\1" correspond à deux caractères identiques consécutifs.
<code>\n</code>	Correspond à <i>n</i> , où <i>n</i> est une valeur d'échappement octale. Les valeurs d'échappement octales doivent comprendre 1, 2 ou 3 chiffres. Par exemple, "\11" et "\011" correspondent tous les deux à un caractère de tabulation. "\0011" équivaut à "\001" & "1". Les valeurs d'échappement octales ne doivent pas excéder 256. Si c'était le cas, seuls les deux premiers chiffres seraient pris en compte dans l'expression. Permet d'utiliser les codes ASCII dans des expressions régulières.
<code>\xn</code>	Correspond à <i>n</i> , où <i>n</i> est une valeur d'échappement hexadécimale. Les valeurs d'échappement hexadécimales doivent comprendre deux chiffres obligatoirement. Par exemple, "\x41" correspond à "A". "\x041" équivaut à "\x04" & "1". Permet d'utiliser les codes ASCII dans des expressions régulières.

Un élément dans un modèle peut être présent en 1 ou plusieurs exemplaires. Considérons quelques exemples autour du symbole `\d` qui représente 1 chiffre :

modèle	signification
<code>\d</code>	un chiffre
<code>\d?</code>	0 ou 1 chiffre
<code>\d*</code>	0 ou davantage de chiffres
<code>\d+</code>	1 ou davantage de chiffres

Exemples de classes .NET

`\d{2}` 2 chiffres  
`\d{3,}` au moins 3 chiffres  
`\d{5,7}` entre 5 et 7 chiffres

Imaginons maintenant le modèle capable de décrire le format attendu pour une chaîne de caractères :

chaîne recherchée	modèle
une date au format jj/mm/aa	<code>\d{2}/\d{2}/\d{2}</code>
une heure au format hh:mm:ss	<code>\d{2}:\d{2}:\d{2}</code>
un nombre entier non signé	<code>\d+</code>
un suite d'espaces éventuellement vide	<code>\s*</code>
un nombre entier non signé qui peut être précédé ou suivi d'espaces	<code>\s*\d+\s*</code>
un nombre entier qui peut être signé et précédé ou suivi d'espaces	<code>\s*[+ -]?\s*\d+\s*</code>
un nombre réel non signé qui peut être précédé ou suivi d'espaces	<code>\s*\d+(\.\d*)?\s*</code>
un nombre réel qui peut être signé et précédé ou suivi d'espaces	<code>\s*[+ -]?\s*\d+(\.\d*)?\s*</code>
une chaîne contenant le mot juste	<code>\bjuste\b</code>

On peut préciser où on recherche le modèle dans la chaîne :

modèle	signification
<code>^modèle</code>	le modèle commence la chaîne
<code>modèle\$</code>	le modèle finit la chaîne
<code>^modèle\$</code>	le modèle commence et finit la chaîne
<code>modèle</code>	le modèle est cherché partout dans la chaîne en commençant par le début de celle-ci.

chaîne recherchée	modèle
une chaîne se terminant par un point d'exclamation	<code>!\$</code>
une chaîne se terminant par un point	<code>\.\$</code>
une chaîne commençant par la séquence //	<code>^//</code>
une chaîne ne comportant qu'un mot éventuellement suivi ou précédé d'espaces	<code>^\s*\w+\s*\$</code>
une chaîne ne comportant deux mot éventuellement suivis ou précédés d'espaces	<code>^\s*\w+\s*\w+\s*\$</code>
une chaîne contenant le mot secret	<code>\bsecret\b</code>

Les sous-ensembles d'un modèle peuvent être "récupérés". Ainsi non seulement, on peut vérifier qu'une chaîne correspond à un modèle particulier mais on peut récupérer dans cette chaîne les éléments correspondant aux sous-ensembles du modèle qui **ont été entourés de parenthèses**. Ainsi si on analyse une chaîne contenant une date jj/mm/aa et si on veut de plus récupérer les éléments jj, mm, aa de cette date on utilisera le modèle `(\d\d)/(\d\d)/(\d\d)`.

### 3.9.1 Vérifier qu'une chaîne correspond à un modèle donné

Un objet de type `Regex` se construit de la façon suivante :

```
Public Sub New(ByVal pattern As String)
```

Le constructeur crée un objet "expression régulière" à partir d'un modèle passé en paramètre (`pattern`). Une fois l'expression régulière modèle construit, on peut la comparer à des chaînes de caractères avec la méthode **IsMatch** :

```
Overloads Public Function IsMatch(ByVal input As String) As Boolean
```

`IsMatch` rend vrai si la chaîne `input` correspond au modèle de l'expression régulière. Voici un exemple :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.Text.RegularExpressions

Module regex1

  Sub Main()
    ' une expression régulière modèle
    Dim modèle1 As String = "^s*\d+\s*$"
    Dim regex1 As New Regex(modèle1)
    ' comparer un exemplaire au modèle
  
```

```

Dim exemplaire1 As String = " 123  "
If regex1.IsMatch(exemplaire1) Then
    affiche(("[" + exemplaire1 + "] correspond au modèle [" + modèle1 + "]"))
Else
    affiche(("[" + exemplaire1 + "] ne correspond pas au modèle [" + modèle1 + "]"))
End If
Dim exemplaire2 As String = " 123a  "
If regex1.IsMatch(exemplaire2) Then
    affiche(("[" + exemplaire2 + "] correspond au modèle [" + modèle1 + "]"))
Else
    affiche(("[" + exemplaire2 + "] ne correspond pas au modèle [" + modèle1 + "]"))
End If
End Sub

' affiche
Sub affiche(ByVal msg As String)
    Console.Out.WriteLine(msg)
End Sub
End Module

```

et les résultats d'exécution :

```

dos>vbc /r:system.dll regex1.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4 pour Microsoft (R) .NET Framework version
1.1.4322.573

dos>regex1
[ 123  ] correspond au modèle [^\s*\d+\s*$]
[ 123a ] ne correspond pas au modèle [^\s*\d+\s*$]

```

## 3.9.2 Trouver tous les éléments d'une chaîne correspondant à un modèle

La méthode **Matches**

```
Overloads Public Function Matches(ByVal input As String) As MatchCollection
```

rend une collection d'éléments de la chaîne *input* correspondant au modèle comme le montre l'exemple suivant :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.Text.RegularExpressions

Module regex2
    Sub Main()
        ' plusieurs occurrences du modèle dans l'exemple
        Dim modèle2 As String = "\d+"
        Dim regex2 As New Regex(modèle2)
        Dim exemplaire3 As String = " 123 456 789 "
        Dim résultats As MatchCollection = regex2.Matches(exemplaire3)
        affiche(("Modèle=[" + modèle2 + "],exemplaire=[" + exemplaire3 + "]"))
        affiche(("Il y a " & résultats.Count & " occurrences du modèle dans l'exemple "))
        Dim i As Integer
        For i = 0 To résultats.Count - 1
            affiche((résultats(i).Value & " en position " & résultats(i).Index))
        Next i
    End Sub

    'affiche
    Sub affiche(ByVal msg As String)
        Console.Out.WriteLine(msg)
    End Sub
End Module

```

La classe **MatchCollection** a une propriété **Count** qui est le nombre d'éléments de la collection. Si *résultats* est un objet *MatchCollection*, *résultats(i)* est l'élément *i* de cette collection et est de type **Match**. Dans notre exemple, *résultats* est l'ensemble des éléments de la chaîne *exemplaire3* correspondant au modèle *modèle2* et *résultats(i)* l'un de ces éléments. La classe *Match* a deux propriétés qui nous intéressent ici :

- **Value** : la valeur de l'objet *Match* donc l'élément correspondant au modèle
- **Index** : la position où l'élément a été trouvé dans la chaîne explorée

Les résultats de l'exécution du programme précédent :

```
dos>vbc /r:system.dll regex2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4p our Microsoft (R) .NET Framework version
1.1.4322.573

dos>regex2
Modèle=[\d+],exemplaire=[ 123 456 789 ]
Il y a 3 occurrences du modèle dans l'exemplaire
123 en position 2
456 en position 7
789 en position 12
```

### 3.9.3 Récupérer des parties d'un modèle

Des sous-ensembles d'un modèle peuvent être "récupérés". Ainsi non seulement, on peut vérifier qu'une chaîne correspond à un modèle particulier mais on peut récupérer dans cette chaîne les éléments correspondant aux sous-ensembles du modèle qui **ont été entourés de parenthèses**. Ainsi si on analyse une chaîne contenant une date jj/mm/aa et si on veut de plus récupérer les éléments jj, mm, aa de cette date on utilisera le modèle `(\d\d)/(\d\d)/(\d\d)`. Examinons l'exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.Text.RegularExpressions

Module regex2
  Sub Main()
    ' capture d'éléments dans le modèle
    Dim modèle3 As String = "(\d\d):(\d\d):(\d\d)"
    Dim regex3 As New Regex(modèle3)
    Dim exemplaire4 As String = "Il est 18:05:49"
    ' vérification modèle
    Dim résultat As Match = regex3.Match(exemplaire4)
    If résultat.Success Then
      ' l'exemplaire correspond au modèle
      affiche("L'exemplaire [" + exemplaire4 + "] correspond au modèle [" + modèle3 + "]")
      ' on affiche les groupes
      Dim i As Integer
      For i = 0 To résultat.Groups.Count - 1
        affiche(("groupes[" & i & "]=[" & résultat.Groups(i).Value & "] en position " &
        résultat.Groups(i).Index))
      Next i
    Else
      ' l'exemplaire ne correspond pas au modèle
      affiche("L'exemplaire[" + exemplaire4 + "] ne correspond pas au modèle [" + modèle3 + "]")
    End If
  End Sub

  'affiche
  Sub affiche(ByVal msg As String)
    Console.Out.WriteLine(msg)
  End Sub
End Module
```

L'exécution de ce programme produit les résultats suivants :

```
dos>vbc /r:system.dll regex3.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4

dos>regex3
L'exemplaire [Il est 18:05:49] correspond au modèle [(\d\d):(\d\d):(\d\d)]
groupes[0]=[18:05:49] en position 7
groupes[1]=[18] en position 7
groupes[2]=[05] en position 10
groupes[3]=[49] en position 13
```

La nouveauté se trouve dans la partie de code suivante :

```
' vérification modèle
Dim résultat As Match = regex3.Match(exemplaire4)
If résultat.Success Then
  ' l'exemplaire correspond au modèle
  affiche("L'exemplaire [" + exemplaire4 + "] correspond au modèle [" + modèle3 + "]")
  ' on affiche les groupes
```

```

Dim i As Integer
For i = 0 To résultat.Groups.Count - 1
    affiche(("groupes[" & i & "]=[" & résultat.Groups(i).Value & "]" en position " &
résultat.Groups(i).Index)
Next i
Else

```

La chaîne *exemplaire4* est comparée au modèle *regex3* au travers de la méthode *Match*. Celle-ci rend un objet *Match* déjà présenté. Nous utilisons ici deux nouvelles propriétés de cette classe :

- **Success** : indique s'il y a eu correspondance
- **Groups** : collection où
  - Groups[0] correspond à la partie de la chaîne correspondant au modèle
  - Groups[i] (i>=1) correspond au groupe de parenthèses n° i

Si résultat est de type *Match*, *résultats.Groups* est de type *GroupCollection* et *résultats.Groups[i]* de type *Group*. La classe *Group* a deux propriétés que nous utilisons ici :

- **Value** : la valeur de l'objet Group qui l'élément correspondant au contenu d'une parenthèse
- **Index** : la position où l'élément a été trouvé dans la chaîne explorée

### 3.9.4 Un programme d'apprentissage

Trouver l'expression régulière qui nous permet de vérifier qu'une chaîne correspond bien à un certain modèle est parfois un véritable défi. Le programme suivant permet de s'entraîner. Il demande un modèle et une chaîne et indique alors si la chaîne correspond ou non au modèle.

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Collections
Imports System.Text.RegularExpressions
Imports Microsoft.VisualBasic

Module regex4
Sub Main()
    ' une expression régulière modèle
    Dim modèle, chaîne As String
    Dim regex As Regex = Nothing
    Dim résultats As MatchCollection
    ' on demande à l'utilisateur les modèles et les exemples à comparer à celui-ci
    Dim terminé1 As Boolean = False
    Do While Not terminé1
        Dim erreur As Boolean = True
        Do While Not terminé1 And erreur
            ' on demande le modèle
            Console.Out.Write("Tapez le modèle à tester ou fin pour arrêter :")
            modèle = Console.In.ReadLine()
            ' fini ?
            If modèle.Trim().ToLower() = "fin" Then
                terminé1 = True
            Else
                ' on crée l'expression régulière
                Try
                    regex = New Regex(modèle)
                    erreur = False
                Catch ex As Exception
                    Console.Error.WriteLine(("Erreur : " + ex.Message))
                End Try
            End If
        Loop
        ' fini ?
        If terminé1 Then Exit Sub
        ' on demande à l'utilisateur les exemples à comparer au modèle
        Dim terminé2 As Boolean = False
        Do While Not terminé2
            Console.Out.Write(("Tapez la chaîne à comparer au modèle [" + modèle + "] ou fin pour arrêter :"))
            chaîne = Console.In.ReadLine()
            ' fini ?
            If chaîne.Trim().ToLower() = "fin" Then
                terminé2 = True
            Else
                ' on fait la comparaison
                résultats = regex.Matches(chaîne)
            End If
        Loop
    End Do
End Sub

```

```

' succès ?
If résultats.Count = 0 Then
    Console.Out.WriteLine("Je n'ai pas trouvé de correspondances")
Else
    ' on affiche les éléments correspondant au modèle
    Dim i As Integer
    For i = 0 To résultats.Count - 1
        Console.Out.WriteLine(("J'ai trouvé la correspondance [" & résultats(i).Value & "] en
position " & résultats(i).Index))
        ' des sous-éléments
        If résultats(i).Groups.Count <> 1 Then
            Dim j As Integer
            For j = 1 To (résultats(i).Groups.Count) - 1
                Console.Out.WriteLine((ControlChars.Tab & "sous-élément [" & résultats(i).Groups(j).Value
& "] en position " & résultats(i).Groups(j).Index))
            Next j
        End If
    Next i
End If
End If
Loop
End Sub

'affiche
Sub affiche(ByVal msg As String)
    Console.Out.WriteLine(msg)
End Sub
End Module

```

Voici un exemple d'exécution :

```

Tapez le modèle à tester ou fin pour arrêter :\d+
Tapez la chaîne à comparer au modèle [\d+] ou fin pour arrêter :123 456 789
J'ai trouvé la correspondance [123] en position 0
J'ai trouvé la correspondance [456] en position 4
J'ai trouvé la correspondance [789] en position 8
Tapez la chaîne à comparer au modèle [\d+] ou fin pour arrêter :fin

Tapez le modèle à tester ou fin pour arrêter :(\d\d):(\d\d)
Tapez la chaîne à comparer au modèle [(\d\d):(\d\d)] ou fin pour arrêter :14:15 abcd 17:18 xyzt
J'ai trouvé la correspondance [14:15] en position 0
    sous-élément [14] en position 0
    sous-élément [15] en position 3
J'ai trouvé la correspondance [17:18] en position 11
    sous-élément [17] en position 11
    sous-élément [18] en position 14
Tapez la chaîne à comparer au modèle [(\d\d):(\d\d)] ou fin pour arrêter :fin

Tapez le modèle à tester ou fin pour arrêter :^\s*\d+\s*$
Tapez la chaîne à comparer au modèle [^\s*\d+\s*$] ou fin pour arrêter : 1456
J'ai trouvé la correspondance [ 1456] en position 0
Tapez la chaîne à comparer au modèle [^\s*\d+\s*$] ou fin pour arrêter :fin

Tapez le modèle à tester ou fin pour arrêter :^\s*(\d+)\s*$
Tapez la chaîne à comparer au modèle [^\s*(\d+)\s*$] ou fin pour arrêter :1456
J'ai trouvé la correspondance [1456] en position 0
    sous-élément [1456] en position 0
Tapez la chaîne à comparer au modèle [^\s*(\d+)\s*$] ou fin pour arrêter :abcd 1
456
Je n'ai pas trouvé de correspondances
Tapez la chaîne à comparer au modèle [^\s*(\d+)\s*$] ou fin pour arrêter :fin

Tapez le modèle à tester ou fin pour arrêter :fin

```

### 3.9.5 La méthode Split

Nous avons déjà rencontré cette méthode dans la classe *String* :

```
Overloads Public Function Split(ByVal ParamArray separator() As Char) As String()
```

La chaîne est vue comme une suite de champs séparés par les caractères présents dans le tableau *separator*. Le résultat est le tableau de ces champs.

Le séparateur de champs de la chaîne est ici un des caractères du tableau *separator*. La méthode *Split* de la classe *Regex* nous permet d'exprimer le séparateur en fonction d'un modèle :

```
Overloads Public Function Split(ByVal input As String) As String()
```

Exemples de classes .NET

La chaîne [input] est décomposée en champs, ceux-ci étant séparés par un séparateur correspondant au modèle de l'objet *Regex* courant. Supposons par exemple que nous ayons dans un fichier texte des lignes de la forme *champ1, champ2, ..., champn*. Les champs sont séparés par une virgule mais celle-ci peut être précédée ou suivie d'espaces. La méthode *Split* de la classe *string* ne convient alors pas. Celle de la méthode *Regex* apporte la solution. Si ligne est la ligne lue, les champs pourront être obtenus par

```
dim champs() as string=new Regex("s*,\s*").Split(ligne)
```

comme le montre l'exemple suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Text.RegularExpressions

Module regex5
Sub Main()
' une ligne
Dim ligne As String = "abc , def , ghi"
' un modèle
Dim modèle As New Regex("\s*,\s*")
' décomposition de ligne en champs
Dim champs As String() = modèle.Split(ligne)
' affichage
Dim i As Integer
For i = 0 To champs.Length - 1
Console.Out.WriteLine(("champs[" & i & "]=[" & champs(i) & "]"))
Next i
End Sub
End Module
```

Les résultats d'exécution :

```
dos>vbc /r:system.dll regex5.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4

dos>regex5
champs[0]=[abc]
champs[1]=[def]
champs[2]=[ghi]
```

## 3.10 Les classes *BinaryReader* et *BinaryWriter*

Les classes **BinaryReader** et **BinaryWriter** servent à lire et écrire des fichiers binaires. Considérons l'application suivante. On veut écrire un programme qui s'appellerait de la façon suivante :

```
// syntaxe pg texte bin
// on lit un fichier texte (texte) et on range son contenu dans un
// fichier binaire
// le fichier texte a des lignes de la forme nom : age
// qu'on rangera dans une structure string, int
```

Le fichier texte a le contenu suivant :

```
paul : 10
helene : 15
jacques : 11
sylvain : 12
```

Le programme est le suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Text.RegularExpressions
Imports System.IO
```

```

' BinaryWriter()
Module bw1
' syntaxe pg texte bin
' on lit un fichier texte (texte) et on range son contenu dans un
' fichier binaire
' le fichier texte a des lignes de la forme nom : age
' qu'on rangera dans une structure string, int

Sub Main(ByVal arguments() As String)
' il faut 2 arguments
Dim nbArgs As Integer = arguments.Length
If nbArgs <> 2 Then
    Console.Error.WriteLine("syntaxe : pg texte binaire")
    Environment.Exit(1)
End If
' ouverture du fichier texte en lecture
Dim input As StreamReader = Nothing
Try
    input = New StreamReader(arguments(0))
Catch ex As Exception
    Console.Error.WriteLine("Erreur d'ouverture du fichier " & arguments(0) & "(" & ex.Message & ")")
    Environment.Exit(2)
End Try
' ouverture du fichier binaire en écriture
Dim output As BinaryWriter = Nothing
Try
    output = New BinaryWriter(New FileStream(arguments(1), FileMode.Create, FileAccess.Write))
Catch ex As Exception
    Console.Error.WriteLine("Erreur d'ouverture du fichier " & arguments(1) & "(" & ex.Message & ")")
    Environment.Exit(3)
End Try
' lecture fichier texte - écriture fichier binaire
' ligne du fichier texte
Dim ligne As String
' le séparateur des champs de la ligne
Dim séparateur As New Regex("\s*:\s*")
' le modèle de l'âge
Dim modAge As New Regex("\s*\d+\s*")
Dim numLigne As Integer = 0
Dim traitementFini As Boolean
Dim champs As String()
ligne = input.ReadLine()
While Not (ligne Is Nothing)
    traitementFini = False
    ' ligne vide ?
    If ligne.Trim() = "" Then
        traitementFini = True
    End If
    ' une ligne de plus
    If Not traitementFini Then
        numLigne += 1
        ' une ligne nom : age
        champs = séparateur.Split(ligne)
        ' il nous faut 2 champs
        If champs.Length <> 2 Then
            Console.Error.WriteLine("La ligne n° " & numLigne & " du fichier " & arguments(0) & " a un
nombre de champs incorrect")
            ' ligne suivante
            traitementFini = True
        End If
    End If
    If Not traitementFini Then
        ' le second champ doit être un entier >=0
        If Not modAge.IsMatch(champs(1)) Then
            Console.Error.WriteLine("La ligne n° " & numLigne & " du fichier " & arguments(0) & " a un âge
incorrect")
            ' ligne suivante
            traitementFini = True
        End If
        ' on écrit les données dans le fichier binaire
        output.Write(champs(0))
        output.Write(Integer.Parse(champs(1)))
    End If
    'ligne suivante
    ligne = input.ReadLine()
End While
' fermeture des fichiers
input.Close()
output.Close()
End Sub

```

Attardons-nous sur les opérations concernant la classe *BinaryWriter* :

- l'objet *BinaryWriter* est ouvert par l'opération

```
output = New BinaryWriter(New FileStream(arguments(1), FileMode.Create, FileAccess.Write))
```

L'argument du constructeur doit être un flux (Stream). Ici c'est un flux construit à partir d'un fichier (FileStream) dont on donne :

- le nom
- l'opération à faire, ici *FileMode.Create* pour créer le fichier
- le type d'accès, ici *FileAccess.Write* pour un accès en écriture au fichier

- l'opération d'écriture

```
' on écrit les données dans le fichier binaire
output.Write(champs(0))
output.Write(Integer.Parse(champs(1)))
```

La classe *BinaryWriter* dispose de différentes méthodes *Write* surchargées pour écrire les différents types de données simples

- l'opération de fermeture du flux

```
output.Close()
```

Les résultats de l'exécution précédente vont nous être donnés par le programme qui suit. Celui-ci accepte également deux arguments :

```
' syntaxe pg bin texte
' on lit un fichier binaire bin et on range son contenu dans un fichier texte (texte)
' le fichier binaire a une structure string, int
' le fichier texte a des lignes de la forme nom : age
```

On fait donc l'opération inverse. On lit un fichier binaire pour créer un fichier texte. Si le fichier texte produit est identique au fichier original on saura que la conversion texte --> binaire --> texte s'est bien passée. Le code est le suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Text.RegularExpressions
Imports System.IO

Module br1
  ' syntaxe pg bin texte
  ' on lit un fichier binaire bin et on range son contenu dans un fichier texte (texte)
  ' le fichier binaire a une structure string, int
  ' le fichier texte a des lignes de la forme nom : age

  Sub Main(ByVal arguments() As String)
    ' il faut 2 arguments
    Dim nbArgs As Integer = arguments.Length
    If nbArgs <> 2 Then
      Console.Error.WriteLine("syntaxe : pg binaire texte")
      Environment.Exit(1)
    End If
    ' ouverture du fichier binaire en lecture
    Dim dataIn As BinaryReader = Nothing
    Try
      dataIn = New BinaryReader(New FileStream(arguments(0), FileMode.Open, FileAccess.Read))
    Catch ex As Exception
      Console.Error.WriteLine("Erreur d'ouverture du fichier " & arguments(0) & "(" & ex.Message & ")")
      Environment.Exit(2)
    End Try
    ' ouverture du fichier texte en écriture
    Dim dataOut As StreamWriter = Nothing
    Try
      dataOut = New StreamWriter(arguments(1))
      dataOut.AutoFlush = True
    Catch ex As Exception
      Console.Error.WriteLine("Erreur d'ouverture du fichier " & arguments(1) & "(" & ex.Message & ")")
    End Try
  End Sub
End Module
```

```

Environment.Exit(3)
End Try
' lecture fichier binaire - écriture fichier texte
Dim nom As String ' nom d'une personne
Dim age As Integer ' son âge
' boucle d'exploitation du fichier binaire
While True
' lecture nom
Try
nom = dataIn.ReadString()
Catch
' fin du fichier
Exit Sub
End Try
' lecture age
Try
age = dataIn.ReadInt32()
Catch
Console.Error.WriteLine("Le fichier " & arguments(0) + " ne semble pas avoir un format correct")
Exit Sub
End Try
' écriture dans fichier texte
dataOut.WriteLine(nom & ":" & age)
System.Console.WriteLine(nom & ":" & age)
End While
' on ferme tout
dataIn.Close()
dataOut.Close()
End Sub
End Module

```

Attardons-nous sur les opérations concernant la classe *BinaryReader* :

- l'objet *BinaryReader* est ouvert par l'opération

```
dataIn = New BinaryReader(New FileStream(arguments(0), FileMode.Open, FileAccess.Read))
```

L'argument du constructeur doit être un flux (Stream). Ici c'est un flux construit à partir d'un fichier (FileStream) dont on donne :

- le nom
- l'opération à faire, ici *FileMode.Open* pour ouvrir un fichier existant
- le type d'accès, ici *FileAccess.Read* pour un accès en lecture au fichier

- l'opération de lecture

```
nom = dataIn.ReadString()
age = dataIn.ReadInt32()
```

La classe *BinaryReader* dispose de différentes méthodes *ReadXX* pour lire les différents types de données simples

- l'opération de fermeture du flux

```
dataIn.Close()
```

Si on exécute les deux programmes à la chaîne transformant *personnes.txt* en *personnes.bin* puis *personnes.bin* en *personnes.txt2* on a :

```

dos>more personnes.txt
paul : 10
helene : 15
jacques : 11
sylvain : 12

dos>more personnes.txt2
paul:10
helene:15
jacques:11
sylvain:12

dos>dir
29/04/2002 18:19          54 personnes.txt
29/04/2002 18:19          44 personnes.bin
29/04/2002 18:20          44 personnes.txt2

```

# 4. Interfaces graphiques avec VB.NET et VS.NET

Nous nous proposons ici de montrer comment construire des interfaces graphiques avec VB.NET. Nous voyons tout d'abord quelles sont les classes de base de la plate-forme .NET qui nous permettent de construire une interface graphique. Nous n'utilisons dans un premier temps aucun outil de génération automatique. Puis nous utiliserons Visual Studio.NET (VS.NET), un outil de développement de Microsoft facilitant le développement d'applications avec les langages .NET et notamment la construction d'interfaces graphiques. La version VS.NET utilisée est la version anglaise.

## 4.1 Les bases des interfaces graphiques

### 4.1.1 Une fenêtre simple

Considérons le code suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Drawing
Imports System.Windows.Forms

' la classe formulaire
Public Class Form1
    Inherits Form

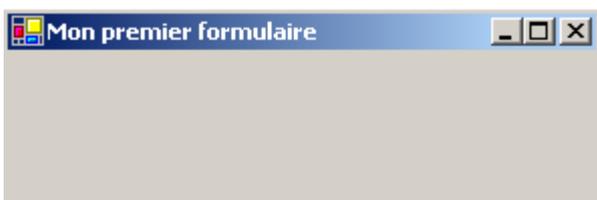
' le constructeur
    Public Sub New()
        ' titre de la fenêtre
        Me.Text = "Mon premier formulaire"
        ' dimensions de la fenêtre
        Me.Size = New System.Drawing.Size(300, 100)
    End Sub

' fonction de test
    Public Shared Sub Main(ByVal args() As String)
        ' on affiche le formulaire
        Application.Run(New Form1)
    End Sub
End Class
```

Le code précédent est compilé puis exécuté

```
dos>vb /r:system.dll /r:system.drawing.dll /r:system.windows.forms.dll frm1.vb
dos>frm1
```

L'exécution affiche la fenêtre suivante :



Une interface graphique dérive en général de la classe de base *System.Windows.Forms.Form* :

```
Public Class Form1
    Inherits Form
```

La classe de base *Form* définit une fenêtre de base avec des boutons de fermeture, agrandissement/réduction, une taille ajustable, ... et gère les événements sur ces objets graphiques. Ici nous spécialisons la classe de base en lui fixant un titre et ses largeur (300) et hauteur (100). Ceci est fait dans son constructeur :

```
Public Sub New()  
    ' titre de la fenêtre  
    Me.Text = "Mon premier formulaire"  
    ' dimensions de la fenêtre  
    Me.Size = New System.Drawing.Size(300, 100)  
End Sub
```

Le titre de la fenêtre est fixée par la propriété *Text* et les dimensions par la propriété *Size*. *Size* est défini dans l'espace de noms *System.Drawing* et est une structure. La procédure *Main* lance l'application graphique de la façon suivante :

```
Application.Run(New Form1)
```

Un formulaire de type *Form1* est créé et affiché, puis l'application se met à l'écoute des événements qui se produisent sur le formulaire (clics, déplacements de souris, ...) et fait exécuter ceux que le formulaire gère. Ici, notre formulaire ne gère pas d'autre événement que ceux gérés par la classe de base *Form* (clics sur boutons fermeture, agrandissement/réduction, changement de taille de la fenêtre, déplacement de la fenêtre, ...).

## 4.1.2 Un formulaire avec bouton

Ajoutons maintenant un bouton à notre fenêtre :

```
' options  
Option Strict On  
Option Explicit On  
  
' espaces de noms  
Imports System  
Imports System.Drawing  
Imports System.Windows.Forms  
  
' la classe formulaire  
Public Class Form1  
    Inherits Form  
  
    ' attributs  
    Private cmdTest As Button  
  
    ' le constructeur  
    Public Sub New()  
        ' le titre  
        Me.Text = "Mon premier formulaire"  
        ' les dimensions  
        Me.Size = New System.Drawing.Size(300, 100)  
        ' un bouton  
        ' création  
        Me.cmdTest = New Button  
        ' position  
        cmdTest.Location = New System.Drawing.Point(110, 20)  
        ' taille  
        cmdTest.Size = New System.Drawing.Size(80, 30)  
        ' libellé  
        cmdTest.Text = "Test"  
        ' gestionnaire d'évt  
        AddHandler cmdTest.Click, AddressOf cmdTest_Click  
        ' ajout bouton au formulaire  
        Me.Controls.Add(cmdTest)  
    End Sub  
  
    ' gestionnaire d'événement  
    Private Sub cmdTest_Click(ByVal sender As Object, ByVal evt As EventArgs)  
        ' il y a eu un clic sur le bouton - on le dit  
        MessageBox.Show("Clic sur bouton", "Clic sur bouton", MessageBoxButtons.OK,  
        MessageBoxIcon.Information)  
    End Sub  
  
    ' fonction de test  
    Public Shared Sub Main(ByVal args() As String)  
        ' on affiche le formulaire  
        Application.Run(New Form1)  
    End Sub  
End Class
```

Nous avons rajouté au formulaire un bouton :

```

' un bouton
' création
Me.cmdTest = New Button
' position
cmdTest.Location = New System.Drawing.Point(110, 20)
' taille
cmdTest.Size = New System.Drawing.Size(80, 30)
' libellé
cmdTest.Text = "Test"
' gestionnaire d'évt
AddHandler cmdTest.Click, AddressOf cmdTest_Click
' ajout bouton au formulaire
Me.Controls.Add(cmdTest)

```

La propriété *Location* fixe les coordonnées (110,20) du point supérieur gauche du bouton à l'aide d'une structure *Point*. Les largeur et hauteur du bouton sont fixées à (80,30) à l'aide d'une structure *Size*. La propriété *Text* du bouton permet de fixer le libellé du bouton. La classe bouton possède un événement Click défini comme suit :

```
Public Event Click As EventHandler
```

où *EventHandler* est fonction "déléguée" ayant la signature suivante :

```
Public Delegate Sub EventHandler(ByVal sender As Object, ByVal e As EventArgs)
```

Cela signifie que le gestionnaire de l'événement [Click] sur le bouton devra avoir la signature du délégué [EventHandler]. Ici, lors d'un clic sur le bouton *cmdTest*, la méthode *cmdTest\_Click* sera appelée. Celle-ci est définie comme suit conformément au modèle *EventHandler* précédent :

```

' gestionnaire d'événement
Private Sub cmdTest_Click(ByVal sender As Object, ByVal evt As EventArgs)
' il y a eu un clic sur le bouton - on le dit
    MessageBox.Show("Clic sur bouton", "Clic sur bouton", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub

```

On se contente d'afficher un message :



La classe est compilée et exécutée :

```

dos>vbcc /r:system.dll /r:system.drawing.dll /r:system.windows.forms.dll frm2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4
dos>frm2

```

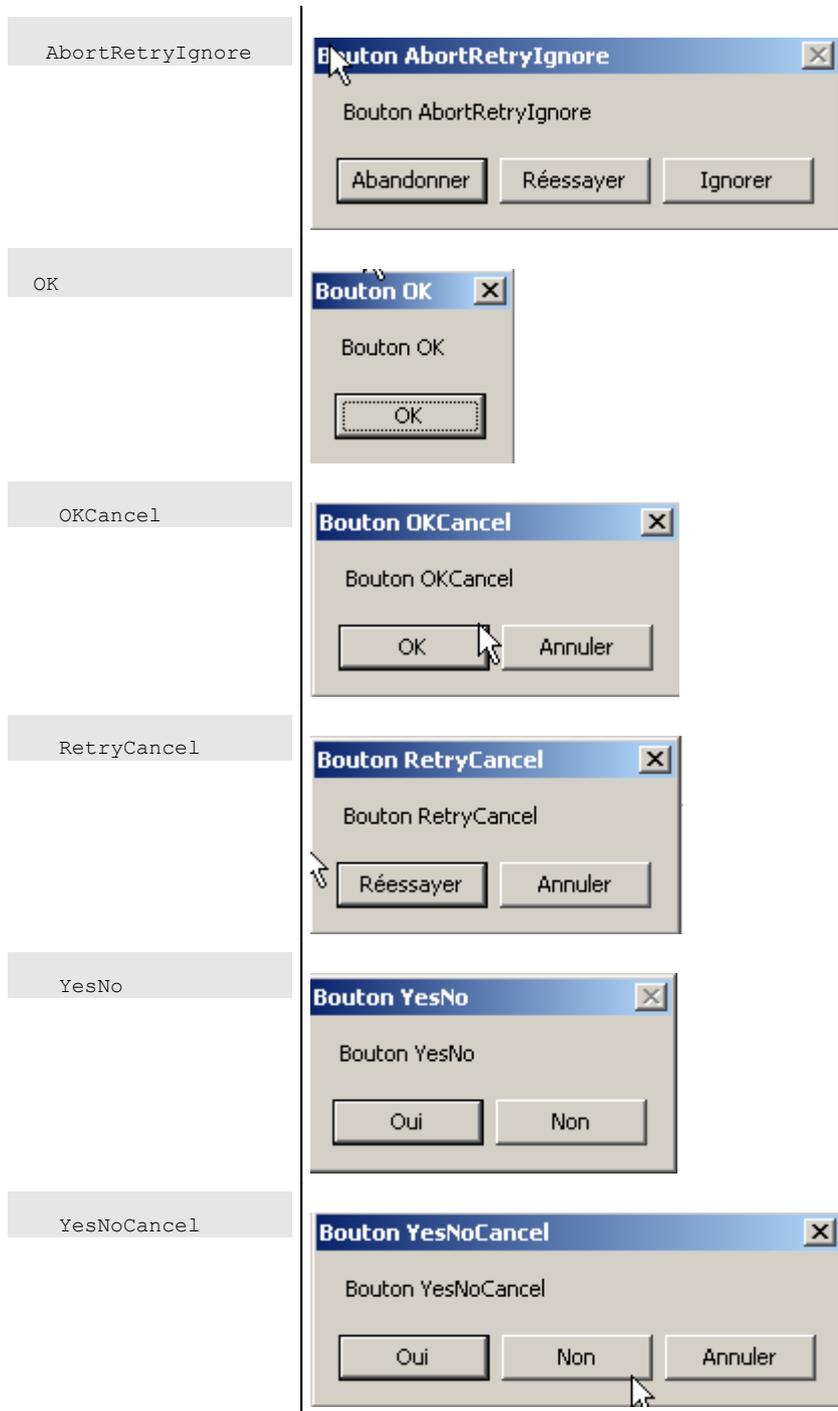
La classe *MessageBox* sert à afficher des messages dans une fenêtre. Nous avons utilisé ici le constructeur

```
Overloads Public Shared Function Show(ByVal owner As IWin32Window, ByVal text As String, ByVal caption As String, ByVal buttons As MessageBoxButtons, ByVal icon As MessageBoxIcon) As DialogResult
```

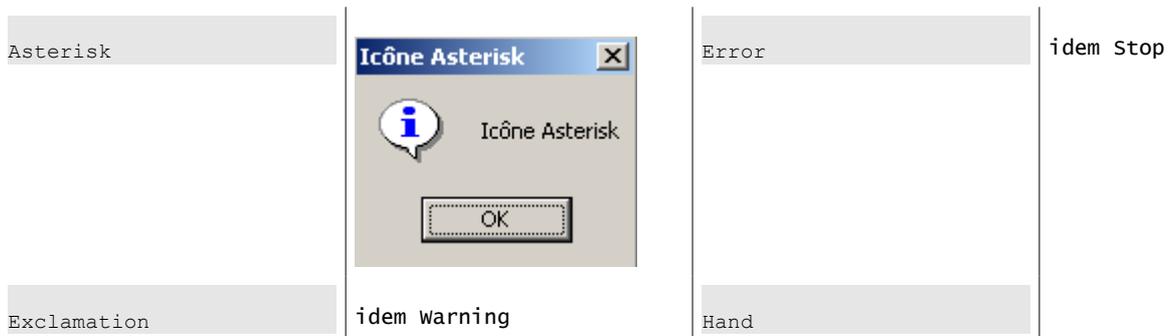
<b>text</b>	le message à afficher
<b>caption</b>	le titre de la fenêtre
<b>buttons</b>	les boutons présents dans la fenêtre
<b>icon</b>	l'icône présente dans la fenêtre

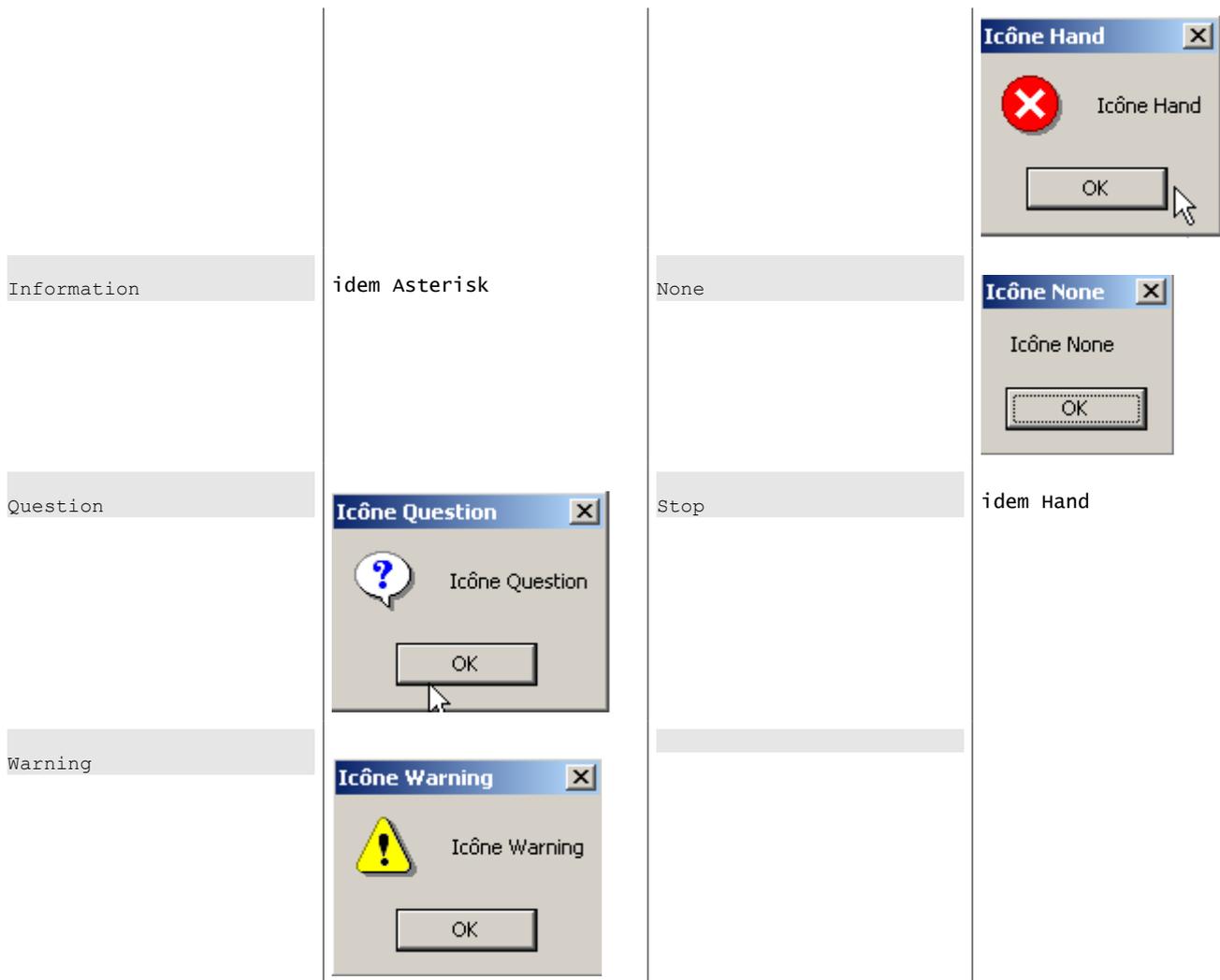
Le paramètre *buttons* peut prendre ses valeurs parmi les constantes suivantes :

constante	boutons



Le paramètre *icon* peut prendre ses valeurs parmi les constantes suivantes :





La méthode *Show* est une méthode statique qui rend un résultat de type `System.Windows.Forms.DialogResult` qui est une énumération :

```
public enum System.Windows.Forms.DialogResult
{
    Abort = 0x00000003,
    Cancel = 0x00000002,
    Ignore = 0x00000005,
    No = 0x00000007,
    None = 0x00000000,
    OK = 0x00000001,
    Retry = 0x00000004,
    Yes = 0x00000006,
}
```

Pour savoir sur quel bouton a appuyé l'utilisateur pour fermer la fenêtre de type *MessageBox* on écrira :

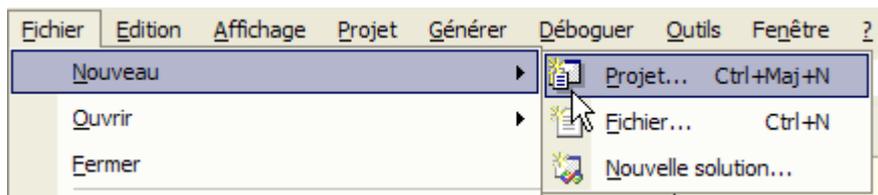
```
dim res as DialogResult=MessageBox.Show(..)
if res=DialogResult.Yes then
    ' il a appuyé sur le bouton oui
    ...
end if
```

## 4.2 Construire une interface graphique avec Visual Studio.NET

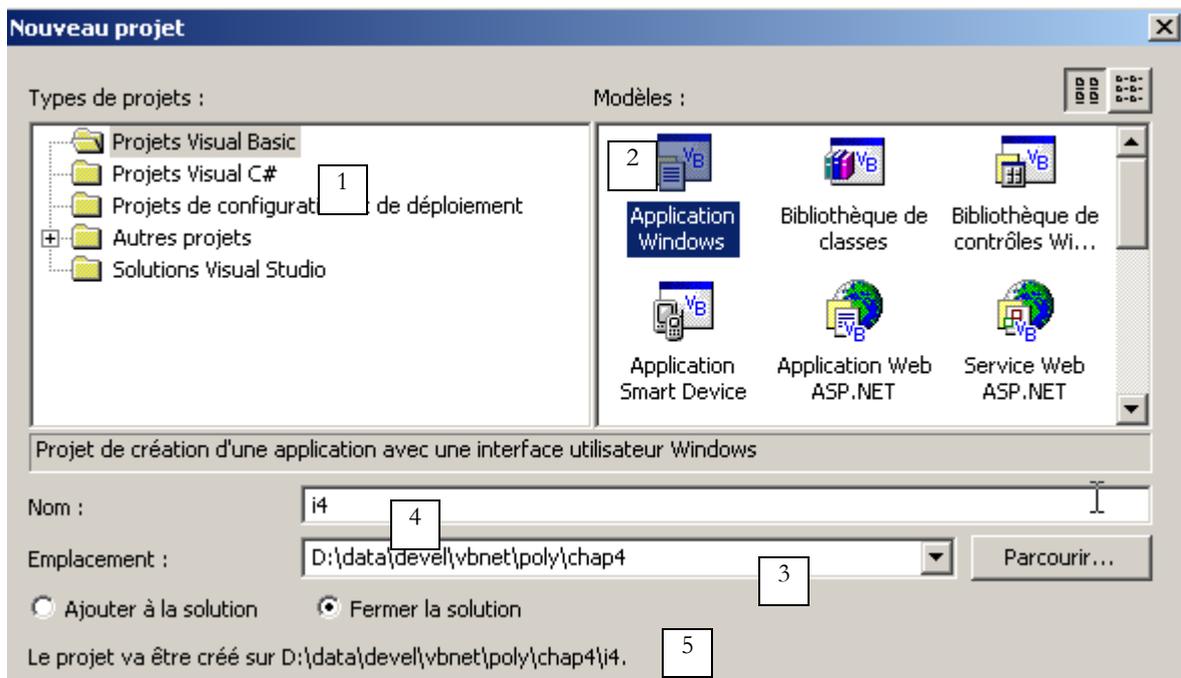
Nous reprenons certains des exemples vus précédemment en les construisant maintenant avec Visual Studio.NET.

### 4.2.1 Création initiale du projet

1. Lancez VS.NET et prendre l'option *Fichier/Nouveau/Projet*



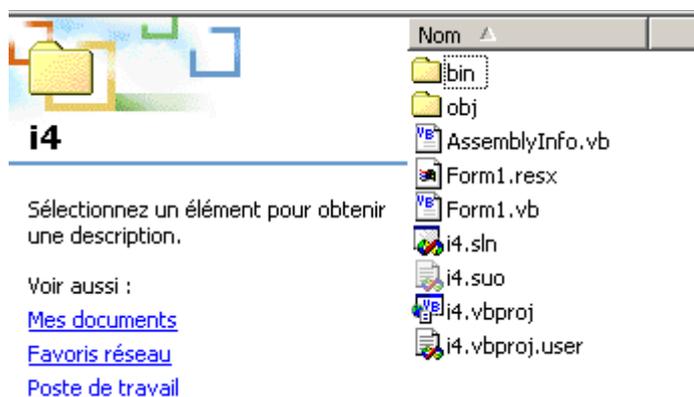
2. donnez les caractéristiques de votre projet



- sélectionnez le type de projet que vous voulez construire, ici un projet VB.NET (1)
- sélectionnez le type d'application que vous voulez construire, ici une application windows (2)
- indiquez dans quel dossier vous voulez placer le sous-dossier du projet (3)
- indiquez le nom du projet (4). Ce sera également le nom du dossier qui contiendra les fichiers du projet
- le nom de ce dossier est rappelé en (5)

3. Un certain nombre de dossiers et de fichiers sont alors créés sous le dossier *i4* :

#### sous-dossiers du dossier projet1

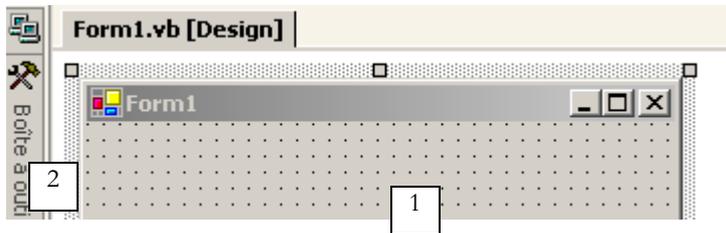


De ces fichiers, seul un est intéressant, le fichier *form1.cs* qui est le fichier source associé au formulaire créé par VS.NET. Nous y reviendrons.

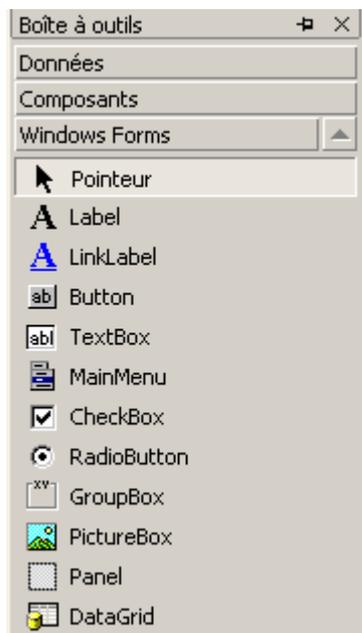
## 4.2.2 Les fenêtrage de l'interface de VS.NET

L'interface de VS.NET laisse maintenant apparaître certains éléments de notre projet *i4* :

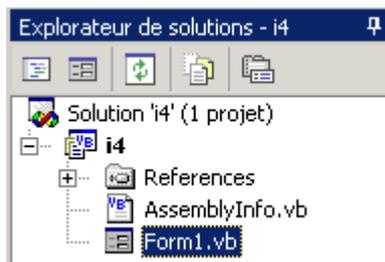
Nous avons une fenêtre de conception de l'interface graphique :



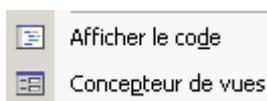
En prenant des contrôles dans la barre d'outils (toolbox 2) et en les déposant sur la surface de la fenêtre (1), nous pouvons construire une interface graphique. Si nous amenons la souris sur la "toolbox" celle-ci s'agrandit et laisse apparaître un certain nombre de contrôles :



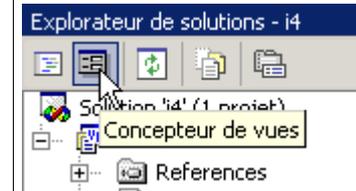
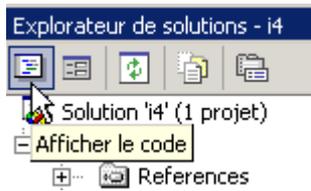
Pour l'instant, nous n'en utilisons aucun. Toujours sur l'écran de VS.NET, nous trouvons la fenêtre de l'explorateur de solutions "Explorer Solution" :



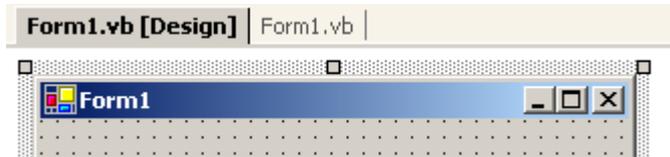
Dans un premier temps, nous ne nous servons peu de cette fenêtre. Elle montre l'ensemble des fichiers formant le projet. Un seul d'entre-eux nous intéresse : le fichier source de notre programme, ici *Form1.vb*. En cliquant droit sur *Form1.vb*, on obtient un menu permettant d'accéder soit au code source de notre interface graphique (*Afficher le code*) soit à l'interface graphique elle-même (*Concepteur de vues*) :



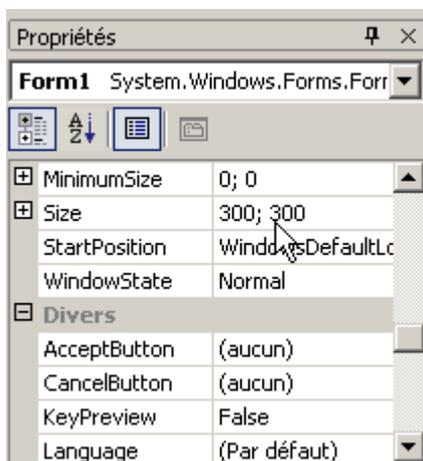
On peut accéder à ces deux entités directement à partir de la fenêtre "Solution Explorer" :



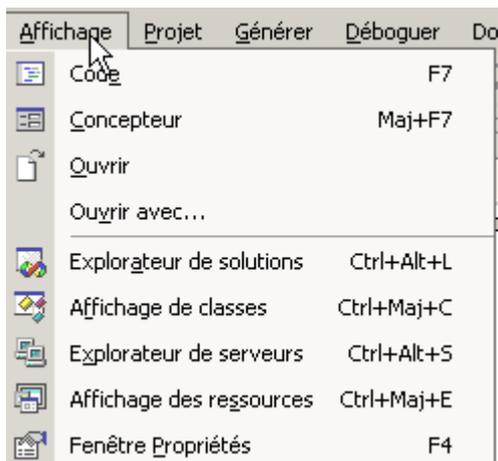
Les fenêtres ouvertes "s'accumulent" dans la fenêtre principale de conception :



Ici *Form1.vb[Design]* désigne la fenêtre de conception et *Form1.vb* la fenêtre de code. Il suffit de cliquer sur l'un des onglets pour passer d'une fenêtre à l'autre. Une autre fenêtre importante présente sur l'écran de VS.NET est la fenêtre des propriétés :



Les propriétés exposées dans la fenêtre sont celles du contrôle actuellement sélectionné dans la fenêtre de conception graphique. On a accès à différentes fenêtres du projet avec le menu *Affichage* :



On y retrouve les fenêtres principales qui viennent d'être décrites ainsi que leurs raccourcis clavier.

### 4.2.3 Exécution d'un projet

Alors même que nous n'avons écrit aucun code, nous avons un projet exécutable. Faites *F5* ou *Débugger/Démarrer* pour l'exécuter. Nous obtenons la fenêtre suivante :



Cette fenêtre peut être agrandie, mise en icône, redimensionnée et fermée.

## 4.2.4 Le code généré par VS.NET

Regardons le code (*Affichage/Code*) de notre application :

```
Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Code généré par le Concepteur Windows Form "

    Public Sub New()
        MyBase.New()

        'Cet appel est requis par le Concepteur Windows Form.
        InitializeComponent()

        'Ajoutez une initialisation quelconque après l'appel InitializeComponent()

    End Sub

    'La méthode substituée Dispose du formulaire pour nettoyer la liste des composants.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Requis par le Concepteur Windows Form
    Private components As System.ComponentModel.IContainer

    'REMARQUE : la procédure suivante est requise par le Concepteur Windows Form
    'Elle peut être modifiée en utilisant le Concepteur Windows Form.
    'Ne la modifiez pas en utilisant l'éditeur de code.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
        Me.Text = "Form2"
    End Sub

#End Region

End Class
```

Une interface graphique dérive de la classe de base *System.Windows.Forms.Form* :

```
Public Class Form1
    Inherits System.Windows.Forms.Form
```

La classe de base *Form* définit une fenêtre de base avec des boutons de fermeture, agrandissement/réduction, une taille ajustable, ... et gère les événements sur ces objets graphiques. Le constructeur du formulaire utilise une méthode *InitializeComponent* dans laquelle les contrôles du formulaire sont créés et initialisés.

```
Public Sub New()
    MyBase.New()

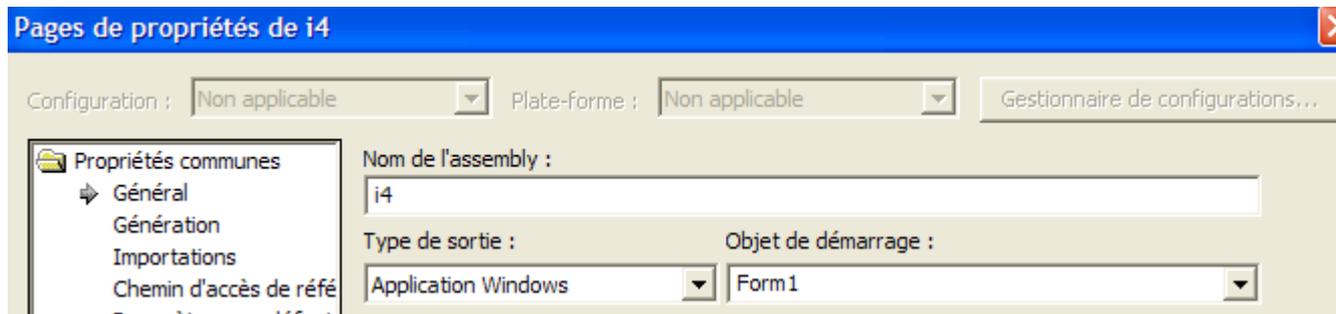
    'Cet appel est requis par le Concepteur Windows Form.
    InitializeComponent()

    'Ajoutez une initialisation quelconque après l'appel InitializeComponent()
End Sub
```

Tout autre travail à faire dans le constructeur peut être fait après l'appel à *InitializeComponent*. La méthode *InitializeComponent*

```
Private Sub InitializeComponent()
    'Form1
    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(292, 53)
    Me.Name = "Form1"
    Me.Text = "Form1"
End Sub
```

fixe le titre de la fenêtre "Form1", sa largeur (292) et sa hauteur (53). Le titre de la fenêtre est fixée par la propriété *Text* et les dimensions par la propriété *Size*. *Size* est défini dans l'espace de noms *System.Drawing* et est une structure. Pour exécuter cette application, il nous faut définir le module principal du projet. Pour cela, nous utilisons l'option [Projets/Propriétés] :



Dans [Objet de démarrage], nous indiquons [Form1] qui est le formulaire que nous venons de créer. Pour lancer l'exécution, nous utilisons l'option [Déboguer/Démarrer] :



## 4.2.5 Compilation dans une fenêtre dos

Maintenant, essayons de compiler et exécuter cette application dans une fenêtre dos :

```
dos>dir form1.vb
14/03/2004 11:53          514 Form1.vb

dos>vbc /r:system.dll /r:system.windows.forms.dll form1.vb
vbc : error BC30420: 'Sub Main' est introuvable dans 'Form1'.
```

Le compilateur indique qu'il ne trouve pas la procédure [Main]. En effet, VB.NET ne l'a pas générée. Nous l'avons cependant déjà rencontrée dans les exemples précédents. Elle a la forme suivante :

```
Shared Sub Main()
    ' on lance l'appli
    Application.Run(New Form1) ' où Form1 est le formulaire
End Sub
```

Rajoutons dans le code de [Form1.vb] le code précédent et recompilons :

```
dos>vbc /r:system.dll /r:system.windows.forms.dll form2.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4

Form2.vb(41) : error BC30451: Le nom 'Application' n'est pas déclaré.

    Application.Run(New Form2)
    ~~~~~
```

Cette fois-ci, le nom [Application] n'est pas connu. Cela veut simplement dire que nous n'avons pas importé son espace de noms [System.Windows.Forms]. Rajoutons l'instruction suivante :

```
Imports System.Windows.Forms

Interfaces graphiques
```

puis recompilons :

```
dos>vbc /r:system.dll /r:system.windows.forms.dll form1.vb
```

Cette fois-ci, ça passe. Exécutons :

```
dos>dir
14/03/2004  11:53                514 Form1.vb
14/03/2004  12:15                3 584 Form1.exe
dos>form1
```



Un formulaire de type *Form1* est créé et affiché. On peut éviter l'ajout de la procédure [Main] en utilisant l'option /m du compilateur qui permet de préciser la classe à exécuter dans le cas où celle-ci hérite de System.Windows.Form :

```
dos>vbc /r:system.dll /r:system.windows.forms.dll /r:system.drawing.dll /m:form2 form2.vb
```

L'option /m:form2 indique que la classe à exécuter est la classe de nom [form2].

## 4.2.6 Gestion des événements

Une fois le formulaire affiché, l'application se met à l'écoute des événements qui se produisent sur le formulaire (clics, déplacements de souris, ...) et fait exécuter ceux que le formulaire gère. Ici, notre formulaire ne gère pas d'autre événement que ceux gérés par la classe de base *Form* (clics sur boutons fermeture, agrandissement/réduction, changement de taille de la fenêtre, déplacement de la fenêtre, ...). Le formulaire généré utilise un attribut *components* qui n'est utilisé nulle part. La méthode *dispose* ne sert également à rien ici. Il en est de même de certains espaces de noms (*Collections*, *ComponentModel*, *Data*) utilisés et de celui défini pour le projet *projet1*. Aussi, dans cet exemple le code peut être simplifié à ce qui suit :

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Public Class Form1
    Inherits System.Windows.Forms.Form

    ' constructeur
    Public Sub New()
        ' construction du formulaire avec ses composants
        InitializeComponent()
    End Sub

    Private Sub InitializeComponent()
        ' taille de la fenêtre
        Me.Size = New System.Drawing.Size(300, 300)
        ' titre de la fenêtre
        Me.Text = "Form1"
    End Sub

    Shared Sub Main()
        ' on lance l'appli
        Application.Run(New Form1)
    End Sub
End Class
```

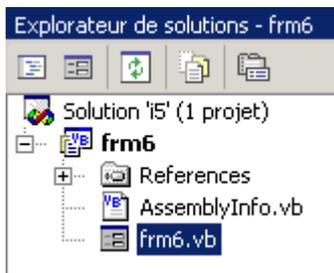
## 4.2.7 Conclusion

Nous accepterons maintenant tel quel le code généré par VS.NET et nous contenterons d'y ajouter le nôtre notamment pour gérer les événements liés aux différents contrôles du formulaire.

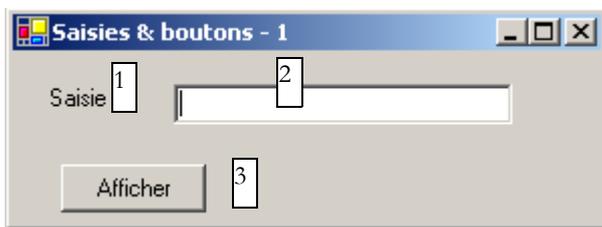
## 4.3 Fenêtre avec champ de saisie, bouton et libellé

### 4.3.1 Conception graphique

Dans l'exemple précédent, nous n'avions pas mis de composants dans la fenêtre. Nous commençons un nouveau projet appelé *interface2*. Pour cela nous suivons la procédure explicitée précédemment pour créer un projet :



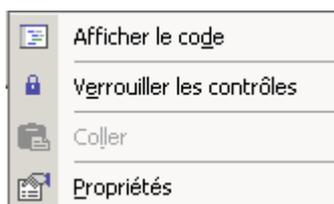
Construisons maintenant une fenêtre avec un bouton, un libellé et un champ de saisie :



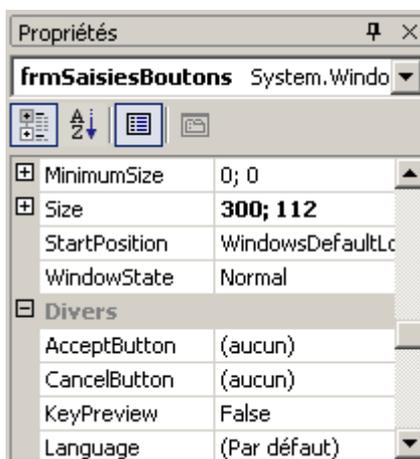
Les champs sont les suivants :

n°	nom	type	rôle
1	lblSaisie	Label	un libellé
2	txtSaisie	TextBox	une zone de saisie
3	btnAfficher	Button	pour afficher dans une boîte de dialogue le contenu de la zone de saisie txtSaisie

On pourra procéder comme suit pour construire cette fenêtre : cliquez droit dans la fenêtre en-dehors de tout composant et choisissez l'option *Propriétés* pour avoir accès aux propriétés de la fenêtre :



La fenêtre de propriétés apparaît alors sur la droite :



Certaines de ces propriétés sont à noter :

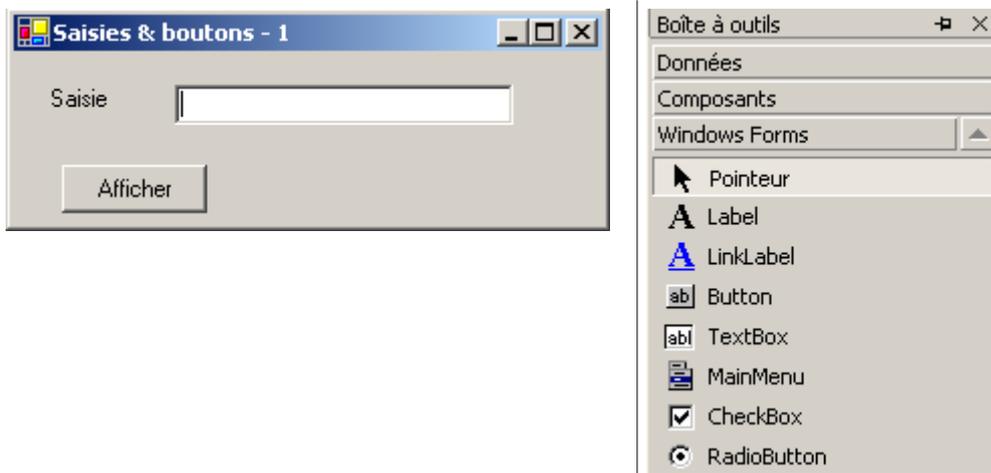
BackColor	pour fixer la couleur de fond de la fenêtre
ForeColor	pour fixer la couleur des dessins ou du texte sur la fenêtre
Menu	pour associer un menu à la fenêtre
Text	pour donner un titre à la fenêtre
FormBorderStyle	pour fixer le type de fenêtre
Font	pour fixer la police de caractères des écritures dans la fenêtre
Name	pour fixer le nom de la fenêtre

Ici, nous fixons les propriétés *Text* et *Name* :

Text	Saisies & boutons - 1
Name	frmSaisiesBoutons

A l'aide de la barre "Boîte à outils"

- sélectionnez les composants dont vous avez besoin
- déposez-les sur la fenêtre et donnez-leur leurs bonnes dimensions



Une fois le composant choisi dans le "toolbox", utilisez la touche "Echap" pour faire disparaître la barre d'outils, puis déposez et dimensionnez le composant. faites-le pour les trois composants nécessaires : *Label*, *TextBox*, *Button*. Pour aligner et dimensionner correctement les composants, utilisez le menu *Format* :



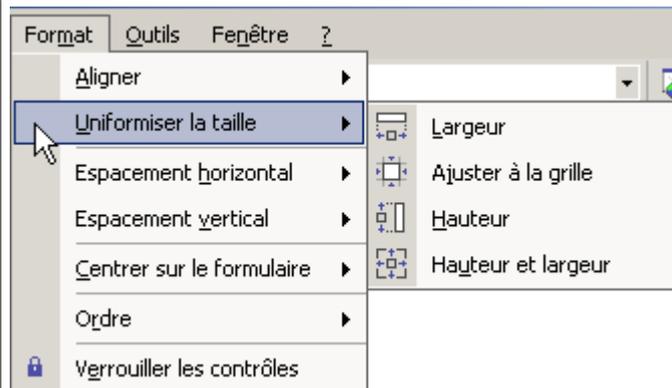
Le principe du formatage est le suivant :

1. sélectionnez les différents composants à formater ensemble (touche Ctrl appuyée)
2. sélectionnez le type de formatage désiré

L'option *Align* vous permet d'aligner des composants

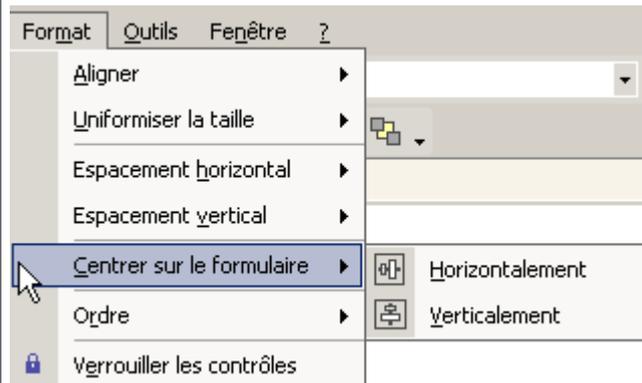


L'option *Make Same Size* permet de faire que des composants aient la même hauteur ou la même largeur :



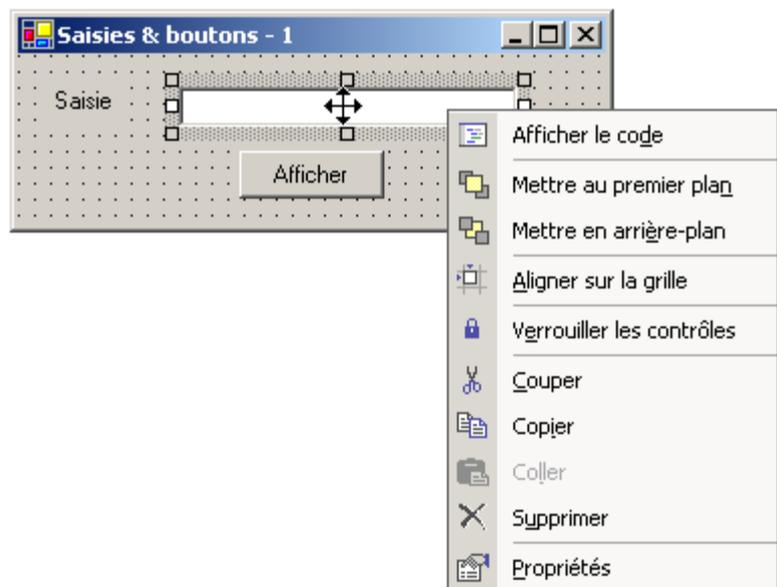
L'option *Horizontal Spacing* permet par exemple d'aligner horizontalement des composants avec des intervalles entre eux de même taille. Idem pour l'option *Vertical Spacing* pour aligner verticalement.

L'option *Center in Form* permet de centrer un composant horizontalement ou verticalement dans la fenêtre :

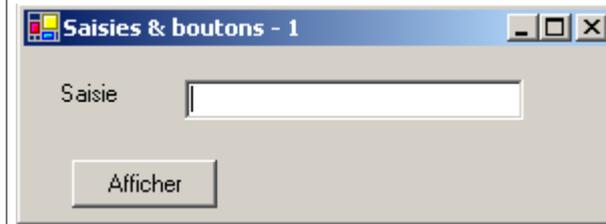


Une fois que les composants sont correctement placés sur la fenêtre, fixez leurs propriétés. Pour cela, cliquez droit sur le composant et prenez l'option *Propriétés* :

- l'étiquette 1 (Label)  
Sélectionnez le composant pour avoir sa fenêtre de propriétés. Dans celle-ci, modifiez les propriétés suivantes : **name** : lblSaisie, **text** : Saisie
- le champ de saisie 2 (TextBox)  
Sélectionnez le composant pour avoir sa fenêtre de propriétés. Dans celle-ci, modifiez les propriétés suivantes : **name** : txtSaisie, **text** : ne rien mettre
- le bouton 3 (Button) : **name** : cmdAfficher, **text** : Afficher
- la fenêtre elle-même : **name** : frmSaisies&Boutons, **text** : Saisies & boutons - 1



Nous pouvons exécuter (ctrl-F5) notre projet pour avoir un premier aperçu de la fenêtre en action :



Fermez la fenêtre. Il nous reste à écrire la procédure liée à un clic sur le bouton *Afficher*.

## 4.3.2 Gestion des événements d'un formulaire

Regardons le code qui a été généré par le concepteur graphique :

```
...
Public Class frmSaisiesBoutons
    Inherits System.Windows.Forms.Form

    ' composants
    Private components As System.ComponentModel.Container = Nothing
    Friend WithEvents btnAfficher As System.Windows.Forms.Button
    Friend WithEvents lblsaisie As System.Windows.Forms.Label
    Friend WithEvents txtsaisie As System.Windows.Forms.TextBox

    ' constructeur
    Public Sub New()
        InitializeComponent()
    End Sub

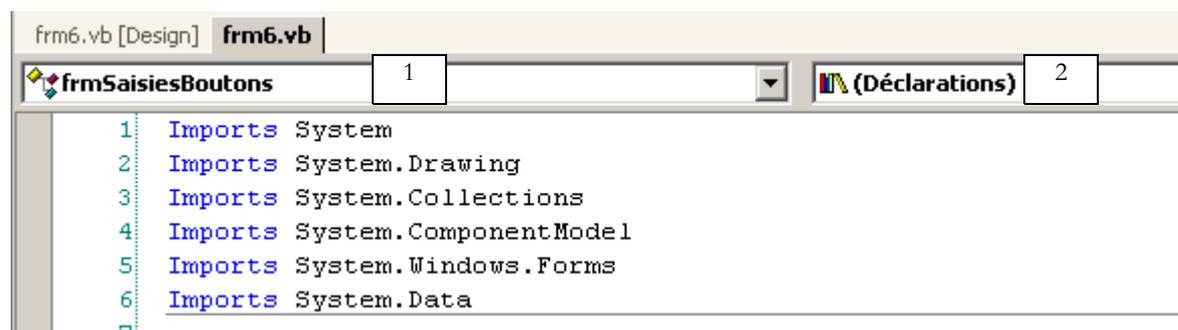
    ...

    ' initialisation des composants
    Private Sub InitializeComponent()
    ...
    End Sub
End Class
```

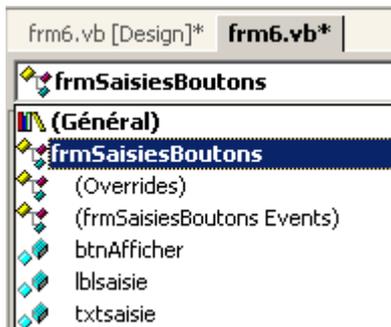
On notera tout d'abord la déclaration particulière des composants :

- le mot clé **Friend** indique une visibilité du composant qui s'étend à toutes les classes du projet
- le mot clé **WithEvents** indique que le composant génère des événements. Nous nous intéressons maintenant à la façon de gérer ces événements

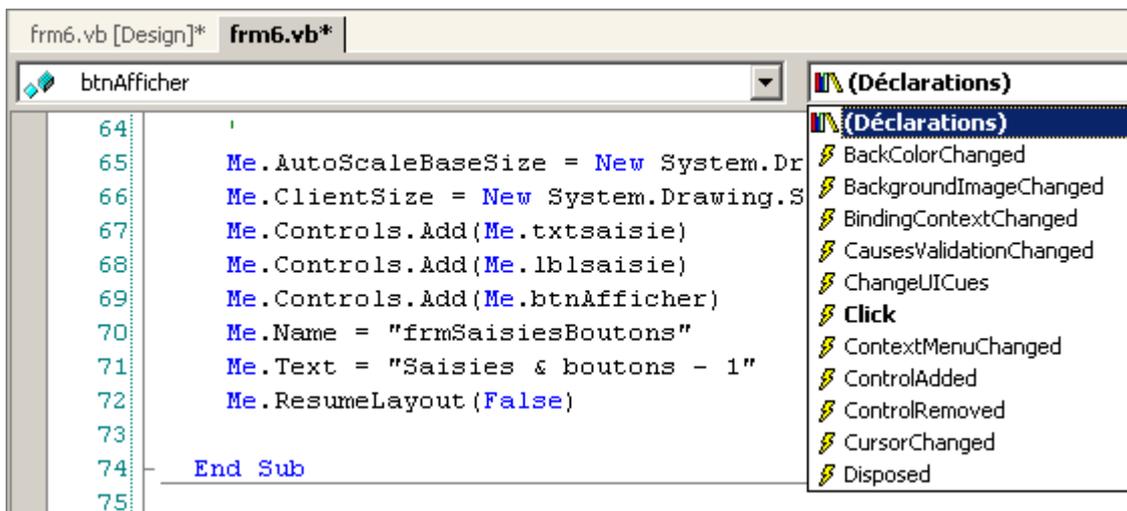
Faites afficher la fenêtre de code du formulaire (Affichage/code ou F7) :



La fenêtre ci-dessus présente deux listes déroulantes (1) et (2). La liste (1) est la liste des composants du formulaire :



La liste(2) la liste des événements associés au composant sélectionné dans (1) :



L'un des événements associés au composant est affiché en gras (ici Click). C'est l'événement par défaut du composant. L'accès au gestionnaire de cet événement particulier peut se faire en double-cliquant sur le composant dans la **fenêtre de conception**. VB.net génère alors automatiquement le squelette du gestionnaire d'événement dans la fenêtre de code et positionne l'utilisateur dessus. L'accès aux gestionnaires des autres événements se fait lui dans la fenêtre de code, en sélectionnant le composant dans la liste (1) et l'événement dans (2). VB.net génère alors le squelette du gestionnaire d'événement ou se positionne dessus s'il était déjà généré :

```
Private Sub btnAfficher_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAfficher.Click
...
End Sub
```

Par défaut, VB.net C\_E nomme le gestionnaire de l'événement E du composant C. On peut changer ce nom si cela nous convient. C'est cependant déconseillé. Les développeurs VB gardent en général le nom généré par VB ce qui donne une cohérence du nommage dans tous les programmes VB. L'association de la procédure **btnAfficher\_Click** à l'événement **Click** du composant **btnAfficher** ne se fait par le nom de la procédure mais par le mot clé **handles** :

```
Handles btnAfficher.Click
```

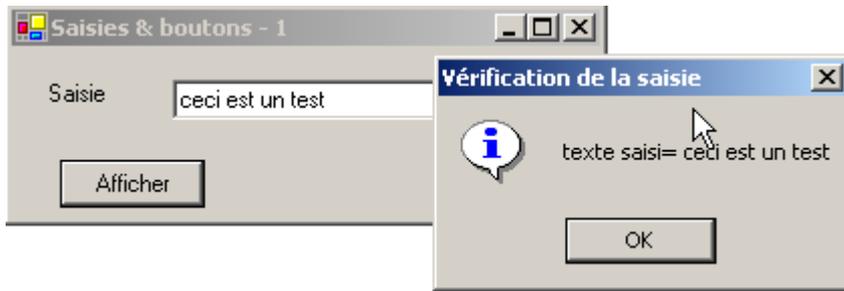
Ci-dessus, le mot clé **handles** précise que la procédure gère l'événement **Click** du composant **btnAfficher**. Le gestionnaire d'événement précédent a deux paramètres :

sender	l'objet à la source de l'événement (ici le bouton)
e	un objet <i>EventArgs</i> qui détaille l'événement qui s'est produit

Nous n'utiliserons aucun de ces paramètres ici. Il ne nous reste plus qu'à compléter le code. Ici, nous voulons présenter une boîte de dialogue avec dedans le contenu du champ *txtSaisie* :

```
Private Sub btnAfficher_Click(ByVal sender As Object, ByVal e As System.EventArgs)
' on affiche le texte qui a été saisi dans la boîte de saisie TxtSaisie
MessageBox.Show("texte saisi= " + txtsaisie.Text, "Vérification de la saisie", MessageBoxButtons.OK,
MessageBoxIcon.Information)
End Sub
```

Si on exécute l'application on obtient la chose suivante :



### 4.3.3 Une autre méthode pour gérer les événements

Pour le bouton `btnAfficher`, VS.NET a généré le code suivant :

```
Private Sub InitializeComponent()
    ...
    'btnAfficher
    Me.btnAfficher.Location = New System.Drawing.Point(102, 48)
    Me.btnAfficher.Name = "btnAfficher"
    Me.btnAfficher.Size = New System.Drawing.Size(88, 24)
    Me.btnAfficher.TabIndex = 2
    Me.btnAfficher.Text = "Afficher"
    ...
End Sub

...

' gestionnaire d'évt clic sur bouton cmdAfficher
Private Sub btnAfficher_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnAfficher.Click
    ...
End Sub
```

Nous pouvons associer la procédure `btnAfficher_Click` à l'événement `Click` du bouton `btnAfficher` d'une autre façon :

```
Private Sub InitializeComponent()
    ...
    'btnAfficher
    Me.btnAfficher.Location = New System.Drawing.Point(102, 48)
    Me.btnAfficher.Name = "btnAfficher"
    Me.btnAfficher.Size = New System.Drawing.Size(88, 24)
    Me.btnAfficher.TabIndex = 2
    Me.btnAfficher.Text = "Afficher"
    AddHandler btnAfficher.Click, AddressOf btnAfficher_Click
    ...
End Sub

...

' gestionnaire d'évt clic sur bouton cmdAfficher
Private Sub btnAfficher_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    ...
End Sub
```

La procédure `btnAfficher_Click` a perdu le mot clé `Handles` perdant ainsi son association avec l'événement `btnAfficher.Click`. Cette association est désormais faite à l'aide du mot clé `AddHandler` :

```
AddHandler btnAfficher.Click, AddressOf btnAfficher_Click
```

Le code ci-dessus qui sera placé dans la procédure `InitializeComponent` du formulaire, associe à l'événement `btnAfficher.Click` la procédure portant le nom `btnAfficher_Click`. Par ailleurs, le composant `btnAfficher` n'a plus besoin du mot clé `WithEvents` :

```
Friend btnAfficher As System.Windows.Forms.Button
```

Quelle est la différence entre les deux méthodes ?

- le mot clé `handles` ne permet d'associer un événement à une procédure **qu'au moment de la conception**. Le concepteur sait à l'avance qu'une procédure P doit gérer les événements E1, E2, ... et il écrit le code

```
Private Sub btnAfficher_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
handles E1, E2, ..., En
```

Il est en effet possible pour une procédure de gérer plusieurs événements.

- le mot clé **addhandler** permet d'associer un événement à une procédure **au moment de l'exécution**. Ceci est utile dans un cadre producteur-consommateur d'événements. Un objet produit un événement particulier susceptible d'intéresser d'autres objets. Ceux-ci s'abonnent auprès du producteur pour recevoir l'événement (une température ayant dépassé un seuil critique, par exemple). Au cours de l'exécution de l'application, le producteur de l'événement sera amené à exécuter différentes instructions :

```
Addhandler E, AddressOf P1
Addhandler E, AddressOf P2
...
Addhandler E, AddressOf Pn
```

où E est l'événement produit par le producteur et Pi des procédures appartenant aux différents objets consommateurs de cet événement. Nous aurons l'occasion de revenir sur une application producteur-consommateur d'événements dans un prochain chapitre.

### 4.3.4 Conclusion

Des deux projets étudiés, nous pouvons conclure qu'une fois l'interface graphique construite avec VS.NET, le travail du développeur consiste à écrire les gestionnaires des événements qu'il veut gérer pour cette interface graphique. Nous ne présenterons désormais que le code de ceux-ci.

## 4.4 Quelques composants utiles

Nous présentons maintenant diverses applications mettant en jeu les composants les plus courants afin de découvrir les principales méthodes et propriétés de ceux-ci. Pour chaque application, nous présentons l'interface graphique et le code intéressant, notamment les gestionnaires d'événements.

### 4.4.1 formulaire Form

Nous commençons par présenter le composant indispensable, le formulaire sur lequel on dépose des composants. Nous avons déjà présenté quelques-unes de ses propriétés de base. Nous nous attardons ici sur quelques événements importants d'un formulaire.

Load	le formulaire est en cours de chargement
Closing	le formulaire est en cours de fermeture
Closed	le formulaire est fermé

L'événement *Load* se produit avant même que le formulaire ne soit affiché. L'événement *Closing* se produit lorsque le formulaire est en cours de fermeture. On peut encore arrêter cette fermeture par programmation. Nous construisons un formulaire de nom *Form1* sans composant :



Nous traitons les trois événements précédents :

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load
' chargement initial du formulaire
MessageBox.Show("Evt Load", "Load")
End Sub

Private Sub Form1_Closing(ByVal sender As Object, ByVal e As System.ComponentModel.CancelEventArgs)
Handles MyBase.Closing
' le formulaire est en train de se fermer
MessageBox.Show("Evt Closing", "Closing")
' on demande confirmation
```

```

    Dim réponse As DialogResult = MessageBox.Show("Voulez-vous vraiment quitter l'application",
"Closing", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
    If réponse = DialogResult.No Then
        e.Cancel = True
    End If
End Sub

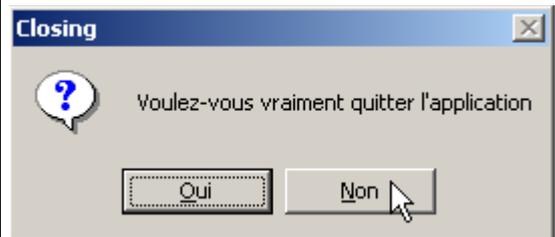
Private Sub Form1_Closed(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Closed
    ' le formulaire est en train de se fermer
    MessageBox.Show("Evt Closed", "Closed")
End Sub

```

Nous utilisons la fonction *MessageBox* pour être averti des différents événements. L'événement *Closing* va se produire lorsque l'utilisateur ferme la fenêtre.



Nous lui demandons alors s'il veut vraiment quitter l'application :

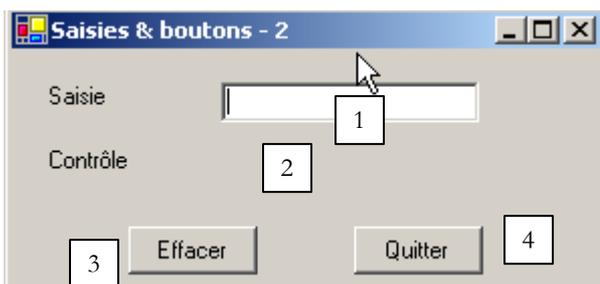


S'il répond Non, nous fixons la propriété *Cancel* de l'événement *CancelEventArgs* *e* que la méthode a reçu en paramètre. Si nous mettons cette propriété à *False*, la fermeture de la fenêtre est abandonnée, sinon elle se poursuit :



## 4.4.2 étiquettes Label et boîtes de saisie TextBox

Nous avons déjà rencontré ces deux composants. *Label* est un composant texte et *TextBox* un composant champ de saisie. Leur propriété principale est *Text* qui désigne soit le contenu du champ de saisie ou le texte du libellé. Cette propriété est en lecture/écriture. L'événement habituellement utilisé pour *TextBox* est *TextChanged* qui signale que l'utilisateur a modifié le champ de saisie. Voici un exemple qui utilise l'événement *TextChanged* pour suivre les évolutions d'un champ de saisie :



n°	type	nom	rôle
1	TextBox	txtSaisie	champ de saisie
2	Label	lblContrôle	affiche le texte de 1 en temps réel
3	Button	cmdEffacer	pour effacer les champs 1 et 2
4	Button	cmdQuitter	pour quitter l'application

Le code pertinent de cette application est celui des gestionnaires d'événements :

```

' clic sur btn quitter
Private Sub cmdQuitter_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles cmdQuitter.Click
    ' clic sur bouton Quitter - on quitte l'application
    Application.Exit()
End Sub

```

```

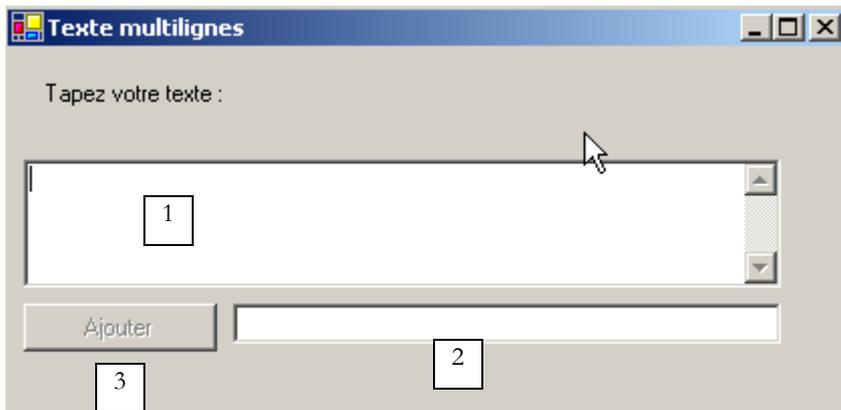
' modification champ txtSaisie
Private Sub txtSaisie_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles txtSaisie.TextChanged
' le contenu du TextBox a changé - on le copie dans le Label lblContrôle
lblContrôle.Text = txtSaisie.Text
End Sub

' clic sur btn effacer
Private Sub cmdEffacer_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles cmdEffacer.Click
' on efface le contenu de la boîte de saisie
txtSaisie.Text = ""
End Sub

' une touche a été tapée
Private Sub txtSaisie_KeyPress(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) _
Handles txtSaisie.KeyPress
' on regarde quelle touche a été tapée
Dim touche As Char = e.KeyChar
If touche = ControlChars.Cr Then
MessageBox.Show(txtSaisie.Text, "Contrôle", MessageBoxButtons.OK, MessageBoxIcon.Information)
e.Handled = True
End If
End Sub

```

On notera la façon de terminer l'application dans la procédure `cmdQuitter_Click` : `Application.Exit()`. L'exemple suivant utilise un `TextBox` multilignes :



La liste des contrôles est la suivante :

n°	type	nom	rôle
1	TextBox	txtMultiLignes	champ de saisie multilignes
2	TextBox	txtAjout	champ de saisie monoligne
3	Button	btnAjouter	Ajoute le contenu de 2 à 1

Pour qu'un `TextBox` devienne multilignes on positionne les propriétés suivantes du contrôle :

<code>Multiline=true</code>	pour accepter plusieurs lignes de texte
<code>ScrollBars=(None, Horizontal, Vertical, Both)</code>	pour demander à ce que le contrôle ait des barres de défilement (Horizontal, Vertical, Both) ou non (None)
<code>AcceptReturn=(True, False)</code>	si égal à true, la touche Entrée fera passer à la ligne
<code>AcceptTab=(True, False)</code>	si égal à true, la touche Tab générera une tabulation dans le texte

Le code utile est celui qui traite le clic sur le bouton [Ajouter] et celui qui traite la modification du champ de saisie [txtAjout] :

```

' évt btnAjouter_Click
Private Sub btnAjouter_Click1(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnAjouter.Click
' ajout du contenu de txtAjout à celui de txtMultilignes
txtMultilignes.Text &= txtAjout.Text
txtAjout.Text = ""
End Sub

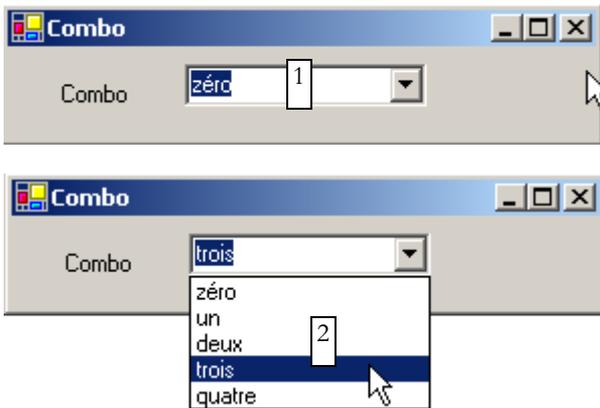
```

```

' evt txtAjout_TextChanged
Private Sub txtAjout_TextChanged1(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles txtAjout.TextChanged
' on fixe l'état du bouton Ajouter
  btnAjouter.Enabled = txtAjout.Text.Trim() <> ""
End Sub

```

### 4.4.3 listes déroulantes ComboBox



Un composant *ComboBox* est une liste déroulante doublée d'une zone de saisie : l'utilisateur peut soit choisir un élément dans (2) soit taper du texte dans (1). Il existe trois sortes de *ComboBox* fixées par la propriété *Style* :

Simple	liste non déroulante avec zone d'édition
DropDown	liste déroulante avec zone d'édition
DropDownList	liste déroulante sans zone d'édition

Par défaut, le type d'un *ComboBox* est *DropDown*. Pour découvrir la classe *ComboBox*, tapez *ComboBox* dans l'index de l'aide (*Aide/Index*). La classe *ComboBox* a un seul constructeur :

```
Public Sub New() | crée un objet ComboBox vide
```

Les éléments du *ComboBox* sont disponibles dans la propriété *Items* :

```
Public ReadOnly Property Items As ComboBox.ObjectCollection
```

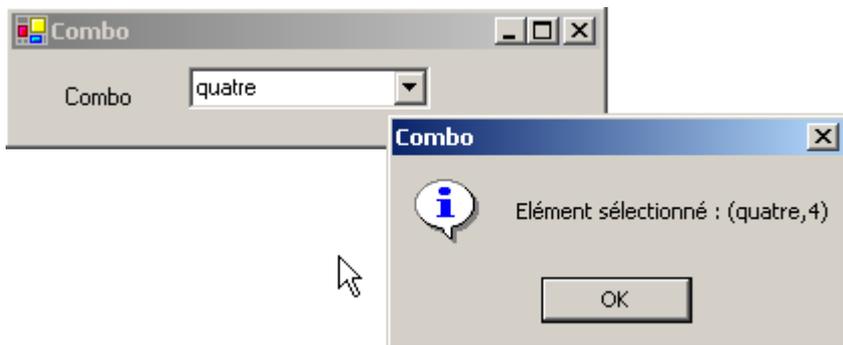
C'est une propriété indexée, *Items(i)* désignant l'élément *i* du Combo. Soit *C* un combo et *C.Items* sa liste d'éléments. On a les propriétés suivantes :

<i>C.Items.Count</i>	nombre d'éléments du combo
<i>C.Items(i)</i>	élément <i>i</i> du combo
<i>C.Add(object o)</i>	ajoute l'objet <i>o</i> en dernier élément du combo
<i>C.AddRange(object() objets)</i>	ajoute un tableau d'objets en fin de combo
<i>C.Insert(int i, object o)</i>	ajoute l'objet <i>o</i> en position <i>i</i> du combo
<i>C.RemoveAt(int i)</i>	enlève l'élément <i>i</i> du combo
<i>C.Remove(object o)</i>	enlève l'objet <i>o</i> du combo
<i>C.Clear()</i>	supprime tous les éléments du combo
<i>C.IndexOf(object o)</i>	rend la position <i>i</i> de l'objet <i>o</i> dans le combo

On peut s'étonner qu'un combo puisse contenir des objets alors qu'habituellement il contient des chaînes de caractères. Au niveau visuel, ce sera le cas. Si un *ComboBox* contient un objet *obj*, il affiche la chaîne *obj.ToString()*. On se rappelle que tout objet à une méthode *ToString* héritée de la classe *object* et qui rend une chaîne de caractères "représentative" de l'objet. L'élément sélectionné

dans le combo C est *C.SelectedItem* ou *C.Items(C.SelectedIndex)* où *C.SelectedIndex* est le n° de l'élément sélectionné, ce n° partant de zéro pour le premier élément.

Lors du choix d'un élément dans la liste déroulante se produit l'événement *SelectedIndexChanged* qui peut être alors utilisé pour être averti du changement de sélection dans le combo. Dans l'application suivante, nous utilisons cet événement pour afficher l'élément qui a été sélectionné dans la liste.



Nous ne présentons que le code pertinent de la fenêtre. Dans le constructeur du formulaire nous remplissons le combo :

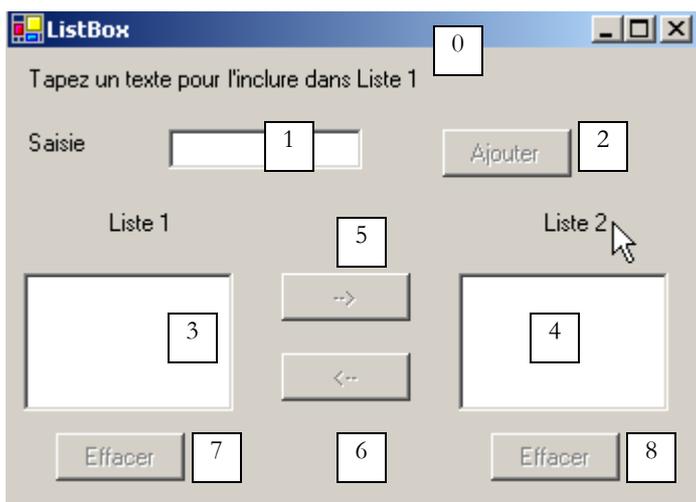
```
Public Sub New()
    ' création formulaire
    InitializeComponent()
    ' remplissage combo
    cmbNombres.Items.AddRange(New String() {"zéro", "un", "deux", "trois", "quatre"})
    ' nous sélectionnons le 1er élément de la liste
    cmbNombres.SelectedIndex = 0
End Sub
```

Nous traitons l'événement *SelectedIndexChanged* du combo qui signale un nouvel élément sélectionné :

```
Private Sub cmbNombres_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles cmbNombres.SelectedIndexChanged
    ' l'élément sélectionné a changé - on l'affiche
    MessageBox.Show("Élément sélectionné : (" & cmbNombres.SelectedItem.ToString & "," &
cmbNombres.SelectedIndex & ")", "Combo", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Sub
```

#### 4.4.4 composant ListBox

On se propose de construire l'interface suivante :



Les composants de cette fenêtre sont les suivants :

n°	type	nom	rôle/propriétés
0	Form	Form1	formulaire - BorderStyle=FixedSingle
1	TextBox	txtSaisie	champ de saisie
2	Button	btnAjouter	bouton permettant d'ajouter le contenu du champ de saisie 1 dans la liste 3

3	ListBox	listBox1	liste 1
4	ListBox	listBox2	liste 2
5	Button	btn1TO2	transfère les éléments sélectionnés de liste 1 vers liste 2
6	Button	cmd2TO1	fait l'inverse
7	Button	btnEffacer1	vide la liste 1
8	Button	btnEffacer2	vide la liste 2

## Fonctionnement

- L'utilisateur tape du texte dans le champ 1. Il l'ajoute à la liste 1 avec le bouton *Ajouter* (2). Le champ de saisie (1) est alors vidé et l'utilisateur peut ajouter un nouvel élément.
- Il peut transférer des éléments d'une liste à l'autre en sélectionnant l'élément à transférer dans l'une des listes et en choisissant le bouton de transfert adéquat 5 ou 6. L'élément transféré est ajouté à la fin de la liste de destination et enlevé de la liste source.
- Il peut double-cliquer sur un élément de la liste 1. Ce élément est alors transféré dans la boîte de saisie pour modification et enlevé de la liste 1.

Les boutons sont allumés ou éteints selon les règles suivantes :

- le bouton *Ajouter* n'est allumé que s'il y a un texte non vide dans le champ de saisie
- le bouton 5 de transfert de la liste 1 vers la liste 2 n'est allumé que s'il y a un élément sélectionné dans la liste 1
- le bouton 6 de transfert de la liste 2 vers la liste 1 n'est allumé que s'il y a un élément sélectionné dans la liste 2
- les boutons 7 et 8 d'effacement des listes 1 et 2 ne sont allumés que si la liste à effacer contient des éléments.

Dans les conditions précédentes, tous les boutons doivent être éteints lors du démarrage de l'application. C'est la propriété *Enabled* des boutons qu'il faut alors positionner à *false*. On peut le faire au moment de la conception ce qui aura pour effet de générer le code correspondant dans la méthode *InitializeComponent* ou de le faire nous-mêmes dans le constructeur comme ci-dessous :

```
Public Sub New()
    ' création initiale du formulaire
    InitializeComponent()
    ' initialisations complémentaires
    ' on inhibe un certain nombre de boutons
    btnAjouter.Enabled = False
    btn1TO2.Enabled = False
    btn2TO1.Enabled = False
    btnEffacer1.Enabled = False
    btnEffacer2.Enabled = False
End Sub
```

L'état du bouton *Ajouter* est contrôlé par le contenu du champ de saisie. C'est l'événement *TextChanged* qui nous permet de suivre les changements de ce contenu :

```
' chgt dans champ txtsaisie
Private Sub txtSaisie_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles txtSaisie.TextChanged
    ' le contenu de txtSaisie a changé
    ' le bouton Ajouter n'est allumé que si la saisie est non vide
    btnAjouter.Enabled = txtSaisie.Text.Trim() <> ""
End Sub
```

L'état des boutons de transfert dépend du fait qu'un élément a été sélectionné ou non dans la liste qu'ils contrôlent :

```
' chgt de l'élément sélectionné sans listBox1
Private Sub listBox1_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles listBox1.SelectedIndexChanged

    ' un élément a été sélectionné
    ' on allume le bouton de transfert 1 vers 2
    btn1TO2.Enabled = True
End Sub

' chgt de l'élément sélectionné sans listBox2
Private Sub listBox2_SelectedIndexChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles listBox2.SelectedIndexChanged

    ' un élément a été sélectionné
    ' on allume le bouton de transfert 2 vers 1
    btn2TO1.Enabled = True
End Sub
```

Le code associé au clic sur le bouton *Ajouter* est le suivant :

```
' clic sur btn Ajouter
```

```

Private Sub btnAjouter_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnAjouter.Click
    ' ajout d'un nouvel élément la liste 1
    listBox1.Items.Add(txtSaisie.Text.Trim())
    ' raz de la saisie
    txtSaisie.Text = ""
    ' Liste 1 n'est pas vide
    btnEffacer1.Enabled = True
    ' retour du focus sur la boîte de saisie
    txtSaisie.Focus()
End Sub

```

On notera la méthode *Focus* qui permet de mettre le "focus" sur un contrôle du formulaire. Le code associé au clic sur les boutons *Effacer*:

```

' clic sur btn Effacer1
Private Sub btnEffacer1_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnEffacer1.Click
    ' on efface la liste 1
    listBox1.Items.Clear()
    btnEffacer1.Enabled = False
End Sub

' clic sur btn effacer2
Private Sub btnEffacer2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    ' on efface la liste 2
    listBox2.Items.Clear()
    btnEffacer2.Enabled = False
End Sub

```

Le code de transfert des éléments sélectionnés d'une liste vers l'autre :

```

' clic sur btn btn1to2
Private Sub btn1TO2_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn1TO2.Click
    ' transfert de l'élément sélectionné dans Liste 1 vers Liste 2
    transfert(listBox1, listBox2)
    ' boutons Effacer
    btnEffacer2.Enabled = True
    btnEffacer1.Enabled = listBox1.Items.Count <> 0
    ' boutons de transfert
    btn1TO2.Enabled = False
    btn2TO1.Enabled = False
End Sub

' clic sur btn btn2to1
Private Sub btn2TO1_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btn2TO1.Click
    ' transfert de l'élément selectionné dans Liste 2 vers Liste 1
    transfert(listBox2, listBox1)
    ' boutons Effacer
    btnEffacer1.Enabled = True
    btnEffacer2.Enabled = listBox2.Items.Count <> 0
    ' boutons de transfert
    btn1TO2.Enabled = False
    btn2TO1.Enabled = False
End Sub

' transfert
Private Sub transfert(ByVal l1 As ListBox, ByVal l2 As ListBox)
    ' transfert de l'élément sélectionné de la liste 1 vers la liste 2
    ' un élément sélectionné ?
    If l1.SelectedIndex = -1 Then Return
    ' ajout dans l2
    l2.Items.Add(l1.SelectedItem)
    ' suppression dans l1
    l1.Items.RemoveAt(l1.SelectedIndex)
End Sub

```

Tout d'abord, on crée une méthode

```
Private Sub transfert(ByVal l1 As ListBox, ByVal l2 As ListBox)
```

qui transfère dans la liste *l2* l'élément sélectionné dans la liste *l1*. Cela nous permet d'avoir une seule méthode au lieu de deux pour transférer un élément de *listBox1* vers *listBox2* ou de *listBox2* vers *listBox1*. Avant de faire le transfert, on s'assure qu'il y a bien un élément sélectionné dans la liste *l1* :

```
' un élément sélectionné ?
```

```
If ll.SelectedIndex = -1 Then Return
```

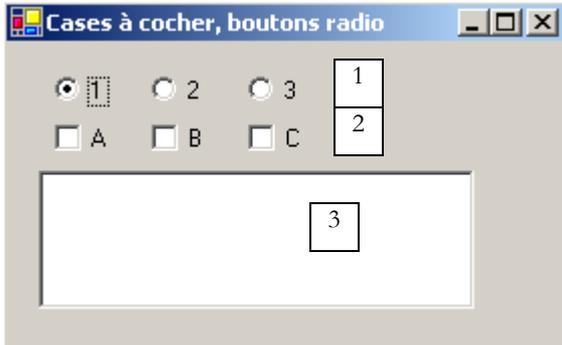
La propriété *SelectedIndex* vaut -1 si aucun élément n'est actuellement sélectionné. Dans les procédures

```
Private Sub btnXTOY_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnXTOY.Click
```

on opère le transfert de la liste X vers la liste Y et on change l'état de certains boutons pour refléter le nouvel état des listes.

## 4.4.5 cases à cocher CheckBox, boutons radio ButtonRadio

Nous nous proposons d'écrire l'application suivante :



Les composants de la fenêtre sont les suivants :

n°	type	nom	rôle
1	RadioButton	radioButton1 radioButton2 radioButton3	3 boutons radio
2	CheckBox	chechBox1 chechBox2 chechBox3	3 cases à cocher
3	ListBox	lstValeurs	une liste

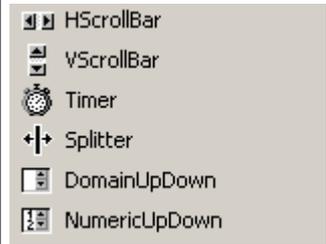
Si on construit les trois boutons radio l'un après l'autre, ils font partie par défaut d'un même groupe. Aussi lorsque l'un est coché, les autres ne le sont pas. L'événement qui nous intéresse pour ces six contrôles est l'événement *CheckChanged* indiquant que l'état de la case à cocher ou du bouton radio a changé. Cet état est représenté dans les deux cas par la propriété booléenne *Check* qui à vrai signifie que le contrôle est coché. Nous avons ici utilisé une seule méthode pour traiter les six événements *CheckChanged*, la méthode *affiche* :

```
' affiche
Private Sub affiche(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles checkBox1.CheckedChanged, checkBox2.CheckedChanged, checkBox3.CheckedChanged, _
radioButton1.CheckedChanged, radioButton2.CheckedChanged, radioButton3.CheckedChanged
' affiche l'état du bouton radio ou de la case à cocher
' est-ce un checkbox ?
If (TypeOf (sender) Is CheckBox) Then
Dim chk As CheckBox = CType(sender, CheckBox)
lstValeurs.Items.Insert(0, chk.Name & "=" & chk.Checked)
End If
' est-ce un radiobutton ?
If (TypeOf (sender) Is RadioButton) Then
Dim rdb As RadioButton = CType(sender, RadioButton)
lstValeurs.Items.Insert(0, rdb.Name & "=" & rdb.Checked)
End If
End Sub
```

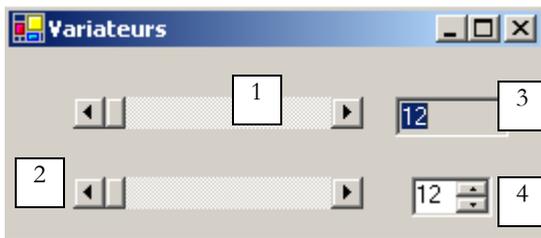
La syntaxe `TypeOf (sender) Is CheckBox` permet de vérifier si l'objet *sender* est de type *CheckBox*. Cela nous permet ensuite de faire un transtypage vers le type exact de *sender*. La méthode *affiche* écrit dans la liste *lstValeurs* le nom du composant à l'origine de l'événement et la valeur de sa propriété *Checked*. A l'exécution, on voit qu'un clic sur un bouton radio provoque deux événements *CheckChanged* : l'un sur l'ancien bouton coché qui passe à "non coché" et l'autre sur le nouveau bouton qui passe à "coché".

## 4.4.6 variateurs ScrollBar

Il existe plusieurs types de variateur : le variateur horizontal (*hScrollBar*), le variateur vertical (*vScrollBar*), l'incrémenteur (*NumericUpDown*).



Réalisons l'application suivante :



n°	type	nom	rôle
1	hScrollBar	hScrollBar1	un variateur horizontal
2	hScrollBar	hScrollBar2	un variateur horizontal qui suit les variations du variateur 1
3	TextBox	txtValeur	affiche la valeur du variateur horizontal ReadOnly=true pour empêcher toute saisie
4	NumericUpDown	incrémenteur	permet de fixer la valeur du variateur 2

- Un variateur *ScrollBar* permet à l'utilisateur de choisir une valeur dans une plage de valeurs entières symbolisée par la "bande" du variateur sur laquelle se déplace un curseur. La valeur du variateur est disponible dans sa propriété **Value**.
- Pour un variateur horizontal, l'extrémité gauche représente la valeur minimale de la plage, l'extrémité droite la valeur maximale, le curseur la valeur actuelle choisie. Pour un variateur vertical, le minimum est représenté par l'extrémité haute, le maximum par l'extrémité basse. Ces valeurs sont représentées par les propriétés **Minimum** et **Maximum** et valent par défaut 0 et 100.
- Un clic sur les extrémités du variateur fait varier la valeur d'un incrément (positif ou négatif) selon l'extrémité cliquée appelée **SmallChange** qui est par défaut 1.
- Un clic de part et d'autre du curseur fait varier la valeur d'un incrément (positif ou négatif) selon l'extrémité cliquée appelée **LargeChange** qui est par défaut 10.
- Lorsqu'on clique sur l'extrémité supérieure d'un variateur vertical, sa valeur diminue. Cela peut surprendre l'utilisateur moyen qui s'attend normalement à voir la valeur "monter". On règle ce problème en donnant une valeur négative aux propriétés *SmallChange* et *LargeChange*.
- Ces cinq propriétés (**Value**, **Minimum**, **Maximum**, **SmallChange**, **LargeChange**) sont accessibles en lecture et écriture.
- L'événement principal du variateur est celui qui signale un changement de valeur : l'événement **Scroll**.

Un composant **NumericUpDown** est proche du variateur : il a lui aussi les propriétés *Minimum*, *Maximum* et *Value*, par défaut 0, 100, 0. Mais ici, la propriété *Value* est affichée dans une boîte de saisie faisant partie intégrante du contrôle. L'utilisateur peut lui même modifier cette valeur sauf si on a mis la propriété *ReadOnly* du contrôle à vrai. La valeur de l'incrément est fixé par la propriété *Increment*, par défaut 1. L'événement principal du composant *NumericUpDown* est celui qui signale un changement de valeur : l'événement **ValueChanged**. Le code utile de notre application est le suivant :

Le formulaire est mis en forme lors de sa construction :

```
' constructeur
Public Sub New()
' création initiale du formulaire
InitializeComponent()
' on donne au variateur 2 les mêmes caractéristiques qu'au variateur 1
hScrollBar2.Minimum = hScrollBar1.Value
hScrollBar2.Minimum = hScrollBar1.Minimum
hScrollBar2.Maximum = hScrollBar1.Maximum
hScrollBar2.LargeChange = hScrollBar1.LargeChange
hScrollBar2.SmallChange = hScrollBar1.SmallChange
' idem pour l'incrémenteur
incrémenteur.Minimum = hScrollBar1.Value
incrémenteur.Minimum = hScrollBar1.Minimum
incrémenteur.Maximum = hScrollBar1.Maximum
incrémenteur.Increment = hScrollBar1.SmallChange
```

```

' on donne au TextBox la valeur du variateur 1
txtValeur.Text = "" & hScrollBar1.Value
End Sub

```

Le gestionnaire qui suit les variations de valeur du variateur 1 :

```

' gestion variateur hscrollbar1
Private Sub hScrollBar1_Scroll(ByVal sender As Object, ByVal e As System.Windows.Forms.ScrollEventArgs)
Handles hScrollBar1.Scroll
' changement de valeur du variateur 1
' on répercute sa valeur sur le variateur 2 et sur le textbox TxtValeur
hScrollBar2.Value = hScrollBar1.Value
txtValeur.Text = "" & hScrollBar1.Value
End Sub

```

Le gestionnaire qui suit les variations de valeur du variateur 2 :

```

' gestion variateur hscrollbar2
Private Sub hScrollBar2_Scroll(ByVal sender As Object, ByVal e As System.Windows.Forms.ScrollEventArgs)
Handles hScrollBar2.Scroll
' on inhibe tout changement du variateur 2
' en le forçant à garder la valeur du variateur 1
e.NewValue = hScrollBar1.Value
End Sub

```

Le gestionnaire qui suit les variations du contrôle *incrémenteur* :

```

' gestion incrémenteur
Private Sub incrémenteur_ValueChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles incrémenteur.ValueChanged
' on fixe la valeur du variateur 2
hScrollBar2.Value = CType(incrémenteur.Value, Integer)
End Sub

```

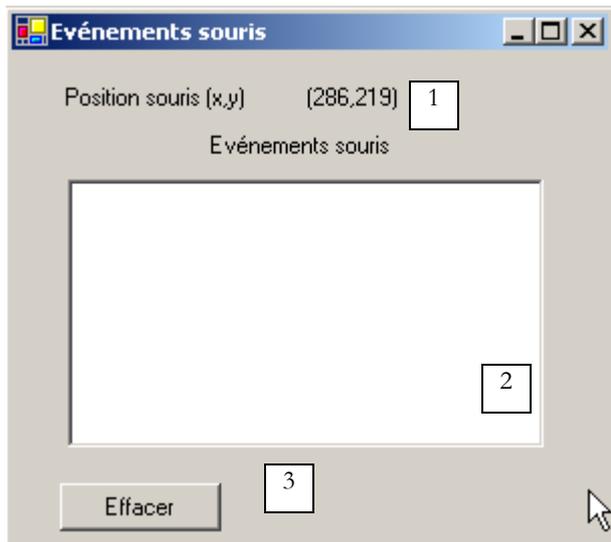
## 4.5 Événements souris

Lorsqu'on dessine dans un conteneur, il est important de connaître la position de la souris pour par exemple afficher un point lors d'un clic. Les déplacements de la souris provoquent des événements dans le conteneur dans lequel elle se déplace.

⚡ <b>MouseDown</b>	⚡ DragDrop
⚡ MouseEnter	⚡ DragEnter
⚡ MouseHover	⚡ DragLeave
⚡ MouseLeave	⚡ DragOver
⚡ <b>MouseMove</b>	
⚡ <b>MouseUp</b>	
⚡ MouseWheel	

MouseEnter	la souris vient d'entrer dans le domaine du contrôle
MouseLeave	la souris vient de quitter le domaine du contrôle
MouseMove	la souris bouge dans le domaine du contrôle
MouseDown	Pression sur le bouton gauche de la souris
MouseUp	Relâchement du bouton gauche de la souris
DragDrop	l'utilisateur lâche un objet sur le contrôle
DragEnter	l'utilisateur entre dans le domaine du contrôle en tirant un objet
DragLeave	l'utilisateur sort du domaine du contrôle en tirant un objet
DragOver	l'utilisateur passe au-dessus domaine du contrôle en tirant un objet

Voici un programme permettant de mieux appréhender à quels moments se produisent les différents événements souris :



n°	type	nom	rôle
1	Label	lblPosition	pour afficher la position de la souris dans le formulaire 1, la liste 2 ou le bouton 3
2	ListBox	lstEvts	pour afficher les évts souris autres que MouseMove
3	Button	btnEffacer	pour effacer le contenu de 2

Les gestionnaires d'événements sont les suivants. Pour suivre les déplacements de la souris sur les trois contrôles, on n'écrit qu'un seul gestionnaire :

```

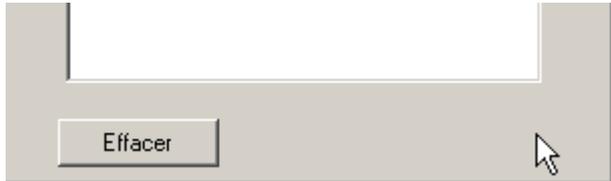
' évt form1_mousemove
Private Sub Form1_MouseMove(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs)
Handles MyBase.MouseMove, lstEvts.MouseMove, btnEffacer.MouseMove, lstEvts.MouseMove
' mvt souris - on affiche les coordonnées (X,Y) de celle-ci
lblPosition.Text = "(" & e.X & "," & e.Y & ")"
End Sub

```

Il faut savoir qu'à chaque fois que la souris entre dans le domaine d'un contrôle, son système de coordonnées change. Son origine (0,0) est le coin supérieur gauche du contrôle sur lequel elle se trouve. Ainsi à l'exécution, lorsqu'on passe la souris du formulaire au bouton, on voit clairement le changement de coordonnées. Afin de mieux voir ces changements de domaine de la souris, on peut utiliser la propriété *Cursor* des contrôles :



Cette propriété permet de fixer la forme du curseur de souris lorsque celle-ci entre dans le domaine du contrôle. Ainsi dans notre exemple, nous avons fixé le curseur à **Default** pour le formulaire lui-même, **Hand** pour la liste 2 et à **No** pour le bouton 3 comme le montrent les copies d'écran ci-dessous.





Dans la méthode [InitializeComponent], le code généré par ces choix est le suivant :

```
Me.lstEvts.Cursor = System.Windows.Forms.Cursors.Hand
Me.btnEffacer.Cursor = System.Windows.Forms.Cursors.No
```

Par ailleurs, pour détecter les entrées et sorties de la souris sur la liste 2, nous traitons les événements **MouseEnter** et **MouseLeave** de cette même liste :

```
' evt lstEvts_MouseEnter
Private Sub lstEvts_MouseEnter(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles lstEvts.MouseEnter
    affiche("MouseEnter sur liste")
End Sub

' evt lstEvts_MouseLeave
Private Sub lstEvts_MouseLeave(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles lstEvts.MouseLeave
    affiche("MouseLeave sur liste")
End Sub

' affiche
Private Sub affiche(ByVal message As String)
    ' on affiche le message en haut de la liste des evts
    lstEvts.Items.Insert(0, message)
End Sub
```



Pour traiter les clics sur le formulaire, nous traitons les événements *MouseDown* et *MouseUp* :

```
' évt Form1_MouseDown
Private Sub Form1_MouseDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) _
Handles MyBase.MouseDown
    affiche("MouseDown sur formulaire")
End Sub

' évt Form1_MouseUp
Private Sub Form1_MouseUp(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) _
Handles MyBase.MouseUp
    affiche("MouseUp sur formulaire")
End Sub
```

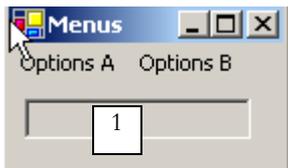


Enfin, le code du gestionnaire de clic sur le bouton *Effacer* :

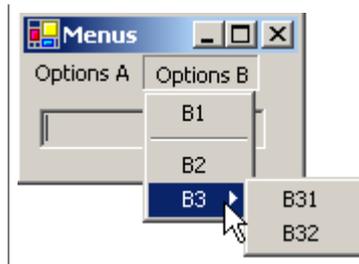
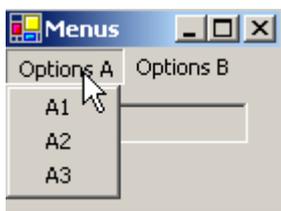
```
' évt btnEffacer_Click
Private Sub btnEffacer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnEffacer.Click
    ' efface la liste des evts
    lstEvts.Items.Clear()
End Sub
```

## 4.6 Créer une fenêtre avec menu

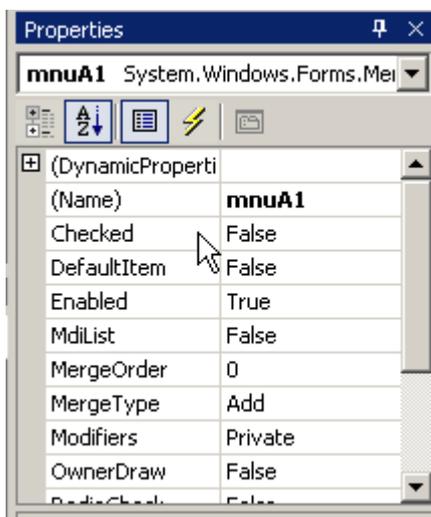
Voyons maintenant comment créer une fenêtre avec menu. Nous allons créer la fenêtre suivante :



Le contrôle 1 est un *TextBox* en lecture seule (*ReadOnly=true*) et de nom *txtStatut*. L'arborescence du menu est la suivante :



Les options de menu sont des contrôles comme les autres composants visuels et ont des propriétés et événements. Par exemple le tableau des propriétés de l'option de menu A1 :



Deux propriétés sont utilisées dans notre exemple :

Name | le nom du contrôle menu  
 Text | le libellé de l'option de menu

Les propriétés des différentes options de menu de notre exemple sont les suivantes :

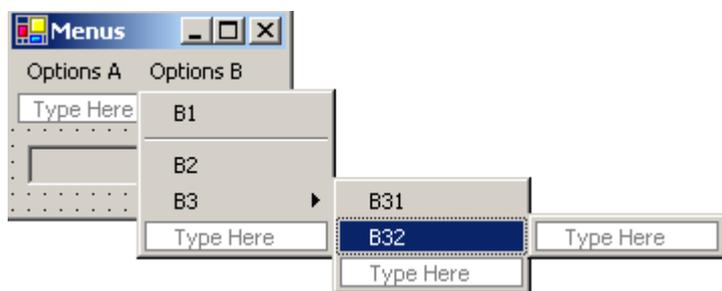
Name	Text
mnuA	options A

mnuA1	A1
mnuA2	A2
mnuA3	A3
mnuB	options B
mnuB1	B1
mnuSep1	- (séparateur)
mnuB2	B2
mnuB3	B3
mnuB31	B31
mnuB32	B32

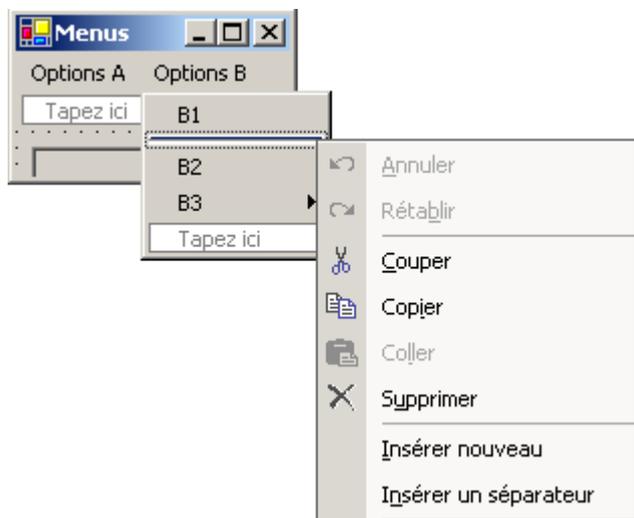
Pour créer un menu, on choisit le composant "MainMenu" dans la barre "ToolBox" :



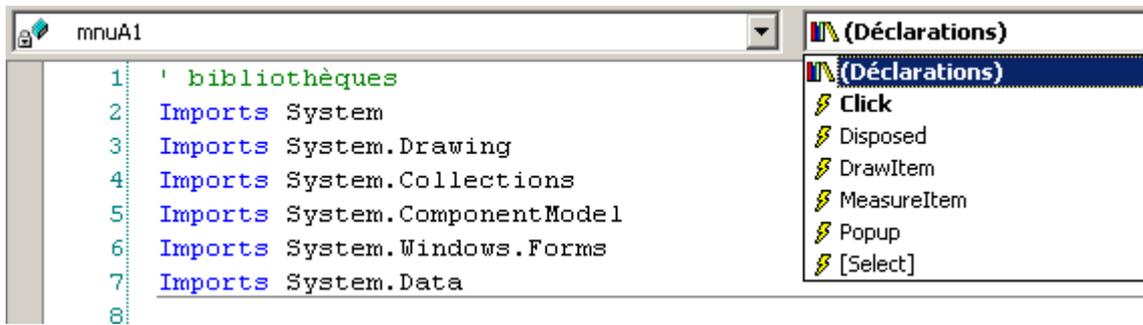
On a alors un menu vide qui s'installe sur le formulaire avec des cases vides intitulées "Type Here". Il suffit d'y indiquer les différentes options du menu :



Pour insérer un séparateur entre deux options comme ci-dessus entre les options B1 et B2, positionnez-vous à l'emplacement du séparateur dans le menu, cliquez droit et prenez l'option *Insert Separator* :



Si on lance l'application par ctrl-F5, on obtient un formulaire avec un menu qui pour l'instant ne fait rien. Les options de menu sont traitées comme des composants : elles ont des propriétés et des événements. Dans [fenêtre de code], sélectionnez le composant mnuA1 puis sélectionnez les événements associés :



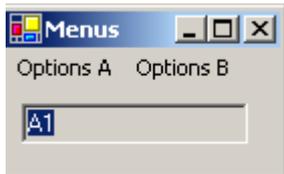
Si ci-dessus on génère l'événement *Click*, VS.NET génère automatiquement la procédure suivante :

```
Private Sub mnuA1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles mnuA.Click
    ....
End Sub
```

Nous pourrions procéder ainsi pour toutes les options de menu. Ici, la même procédure peut être utilisée pour toutes les options. Aussi renomme-t-on la procédure précédente *affiche* et nous déclarons les événements qu'elle gère :

```
Private Sub affiche(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles mnuA1.Click, mnuA2.Click, mnuB1.Click, mnuB2.Click, mnuB31.Click, mnuB32.Click
    ' affiche dans le TextBox le nom du sous-menu choisi
    txtStatut.Text = (CType(sender, MenuItem)).Text
End Sub
```

Dans cette méthode, nous nous contentons d'afficher la propriété *Text* de l'option de menu à la source de l'événement. La source de l'événement *sender* est de type *object*. Les options de menu sont elles de type *MenuItem*, aussi est-on obligé ici de faire un transtypage de *object* vers *MenuItem*. Exécutez l'application et sélectionnez l'option A1 pour obtenir le message suivant :



Le code utile de cette application, outre celui de la méthode *affiche*, est celui de la construction du menu dans le constructeur du formulaire (*InitializeComponent*) :

```
Private Sub InitializeComponent ()
    Me.mainMenu1 = New System.Windows.Forms.MainMenu
    Me.mnuA = New System.Windows.Forms.MenuItem
    Me.mnuA1 = New System.Windows.Forms.MenuItem
    Me.mnuA2 = New System.Windows.Forms.MenuItem
    Me.mnuA3 = New System.Windows.Forms.MenuItem
    Me.mnuB = New System.Windows.Forms.MenuItem
    Me.mnuB1 = New System.Windows.Forms.MenuItem
    Me.mnuB2 = New System.Windows.Forms.MenuItem
    Me.mnuB3 = New System.Windows.Forms.MenuItem
    Me.mnuB31 = New System.Windows.Forms.MenuItem
    Me.mnuB32 = New System.Windows.Forms.MenuItem
    Me.txtStatut = New System.Windows.Forms.TextBox
    Me.mnuSep1 = New System.Windows.Forms.MenuItem
    Me.SuspendLayout ()
    '
    ' mainMenu1
    '
    Me.mainMenu1.MenuItems.AddRange (New System.Windows.Forms.MenuItem () {Me.mnuA, Me.mnuB})
    '
    ' mnuA
    '
    Me.mnuA.Index = 0
    Me.mnuA.MenuItems.AddRange (New System.Windows.Forms.MenuItem () {Me.mnuA1, Me.mnuA2, Me.mnuA3})
    Me.mnuA.Text = "Options A"
    '
    ' mnuA1
    '
    Me.mnuA1.Index = 0
    Me.mnuA1.Text = "A1"
    '
End Sub
```

```

' mnuA2
'
Me.mnuA2.Index = 1
Me.mnuA2.Text = "A2"
'
' mnuA3
'
Me.mnuA3.Index = 2
Me.mnuA3.Text = "A3"
'
' mnuB
'
Me.mnuB.Index = 1
Me.mnuB.MenuItems.AddRange(New System.Windows.Forms.MenuItem() {Me.mnuB1, Me.mnuSep1, Me.mnuB2,
Me.mnuB3})

Me.mnuB.Text = "Options B"
'
' mnuB1
'
Me.mnuB1.Index = 0
Me.mnuB1.Text = "B1"
'
' mnuB2
'
Me.mnuB2.Index = 2
Me.mnuB2.Text = "B2"

'
' mnuB3
'
Me.mnuB3.Index = 3
Me.mnuB3.MenuItems.AddRange(New System.Windows.Forms.MenuItem() {Me.mnuB31, Me.mnuB32})
Me.mnuB3.Text = "B3"
'
' mnuB31
'
Me.mnuB31.Index = 0
Me.mnuB31.Text = "B31"
'
' mnuB32
'
Me.mnuB32.Index = 1
Me.mnuB32.Text = "B32"
'
' txtStatut
'
Me.txtStatut.Location = New System.Drawing.Point(8, 8)
Me.txtStatut.Name = "txtStatut"
Me.txtStatut.ReadOnly = True
Me.txtStatut.Size = New System.Drawing.Size(112, 20)
Me.txtStatut.TabIndex = 0
Me.txtStatut.Text = ""
'
' mnuSep1
'
Me.mnuSep1.Index = 1
Me.mnuSep1.Text = "-"
'
' Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(136, 42)
Me.Controls.Add(txtStatut)
Me.Menu = Me.mainMenu1
Me.Name = "Form1"
Me.Text = "Menus"
Me.ResumeLayout(False)
End Sub

```

On notera l'instruction qui associe le menu au formulaire :

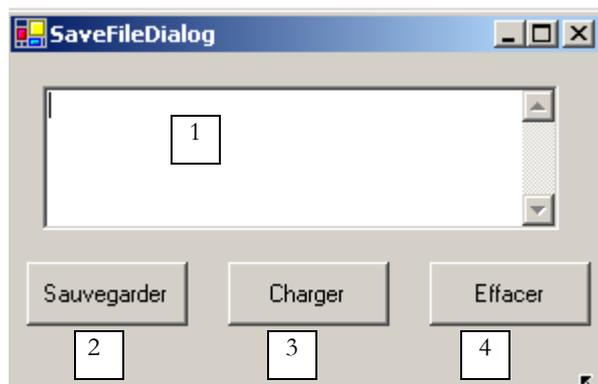
```
Me.Menu = Me.mainMenu1
```

## 4.7 Composants non visuels

Nous nous intéressons maintenant à un certain nombre de composants non visuels : on les utilise lors de la conception mais on ne les voit pas lors de l'exécution.

## 4.7.1 Boîtes de dialogue OpenFileDialog et SaveFileDialog

Nous allons construire l'application suivante :



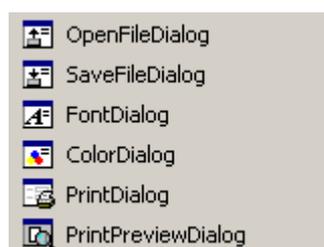
Les contrôles sont les suivants :

N°	type	nom	rôle
1	TextBox multilignes	txtTexte	texte tapé par l'utilisateur ou chargé à partir d'un fichier
2	Button	btnSauvegarde	permet de sauvegarder le texte de 1 dans un fichier texte
3	Button	btnCharger	permet de charger le contenu d'un fichier texte dans 1
4	Button	btnEffacer	efface le contenu de 1

Deux contrôles non visuels sont utilisés :



Lorsqu'ils sont pris dans le "ToolBox " et déposés sur le formulaire, ils sont placés dans une zone à part en bas du formulaire. Les composants "Dialog" sont pris dans le "ToolBox" :



Le code du bouton *Effacer* est simple :

```
Private Sub btnEffacer_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnEffacer.Click
' on efface la boîte de saisie
txtTexte.Text = ""
End Sub
```

La classe **SaveFileDialog** est définie comme suit :

**Bibliothèque de classes .NET Framework**  
**SaveFileDialog, classe** [Visual Basic]

Représente une boîte de dialogue commune qui permet à l'utilisateur de spécifier les options d'enregistrement d'un fichier. Cette classe ne peut pas être héritée.

Pour obtenir une liste de tous les membres de ce type, consultez [SaveFileDialog, membres](#).

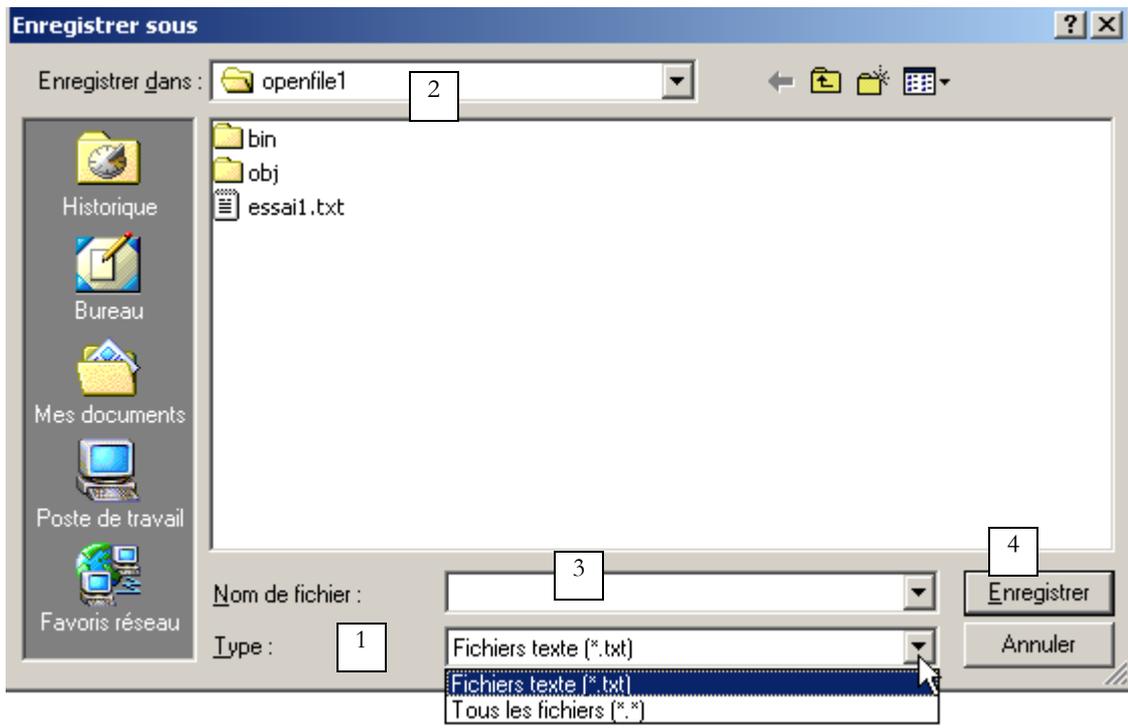
- [System.Object](#)
- [System.MarshalByRefObject](#)
- [System.ComponentModel.Component](#)
- [System.Windows.Forms.CommonDialog](#)
- [System.Windows.Forms.FileDialog](#)
- System.Windows.Forms.SaveFileDialog**

```
NotInheritable Public Class SaveFileDialog
    Inherits FileDialog
```

Elle dérive de plusieurs niveaux de classe. De ces nombreuses propriétés et méthodes nous retiendrons les suivantes :

string Filter	les types de fichiers proposés dans la liste déroulante des types de fichiers de la boîte de dialogue
int FilterIndex	le n° du type de fichier proposé par défaut dans la liste ci-dessus. Commence à 0.
string InitialDirectory	le dossier présenté initialement pour la sauvegarde du fichier
string FileName	le nom du fichier de sauvegarde indiqué par l'utilisateur
DialogResult.ShowDialog()	méthode qui affiche la boîte de dialogue de sauvegarde. Rend un résultat de type DialogResult.

La méthode *ShowDialog* affiche une boîte de dialogue analogue à la suivante :



- 1 | liste déroulante construite à partir de la propriété **Filter**. Le type de fichier proposé par défaut est fixé par **FilterIndex**
- 2 | dossier courant, fixé par **InitialDirectory** si cette propriété a été renseignée
- 3 | nom du fichier choisi ou tapé directement par l'utilisateur. Sera disponible dans la propriété **FileName**
- 4 | boutons **Enregistrer/Annuler**. Si le bouton *Enregistrer* est utilisé, la fonction *ShowDialog* rend le résultat **DialogResult.OK**

La procédure de sauvegarde peut s'écrire ainsi :

```
Private Sub btnSauvegarder_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles btnSauvegarder.Click
```

```

' on sauvegarde la boîte de saisie dans un fichier texte
' on paramètre la boîte de dialogue saveFileDialog1
saveFileDialog1.InitialDirectory = Application.ExecutablePath
saveFileDialog1.Filter = "Fichiers texte (*.txt)|*.txt|Tous les fichiers (*.*)|*.*"
saveFileDialog1.FilterIndex = 0
' on affiche la boîte de dialogue et on récupère son résultat
If saveFileDialog1.ShowDialog() = DialogResult.OK Then
    ' on récupère le nom du fichier
    Dim nomFichier As String = saveFileDialog1.FileName
    Dim fichier As StreamWriter = Nothing
    Try
        ' on ouvre le fichier en écriture
        fichier = New StreamWriter(nomFichier)
        ' on écrit le texte dedans
        fichier.Write(txtTexte.Text)
    Catch ex As Exception
        ' problème
        MessageBox.Show("Problème à l'écriture du fichier (" + ex.Message + ")", "Erreur",
        MessageBoxButtons.OK, MessageBoxIcon.Error)
    Return
    Finally
        ' on ferme le fichier
        Try
            fichier.Close()
        Catch
        End Try
    End Try
End If
End Sub

```

- On fixe le dossier initial au dossier qui contient l'exécutable de l'application :

```
saveFileDialog1.InitialDirectory = Application.ExecutablePath
```

- On fixe les types de fichiers à présenter

```
saveFileDialog1.Filter = "Fichiers texte (*.txt)|*.txt|Tous les fichiers (*.*)|*.*"
```

On notera la syntaxe des filtres filtre1 | filtre2 | .. | filtren avec filtrei = Texte | modèle de fichier. Ici l'utilisateur aura le choix entre les fichiers \*.txt et \*.\*.

- On fixe le type de fichier à présenter au début

```
saveFileDialog1.FilterIndex = 0
```

Ici, ce sont les fichiers de type \*.txt qui seront présentés tout d'abord à l'utilisateur.

- La boîte de dialogue est affichée et son résultat récupéré

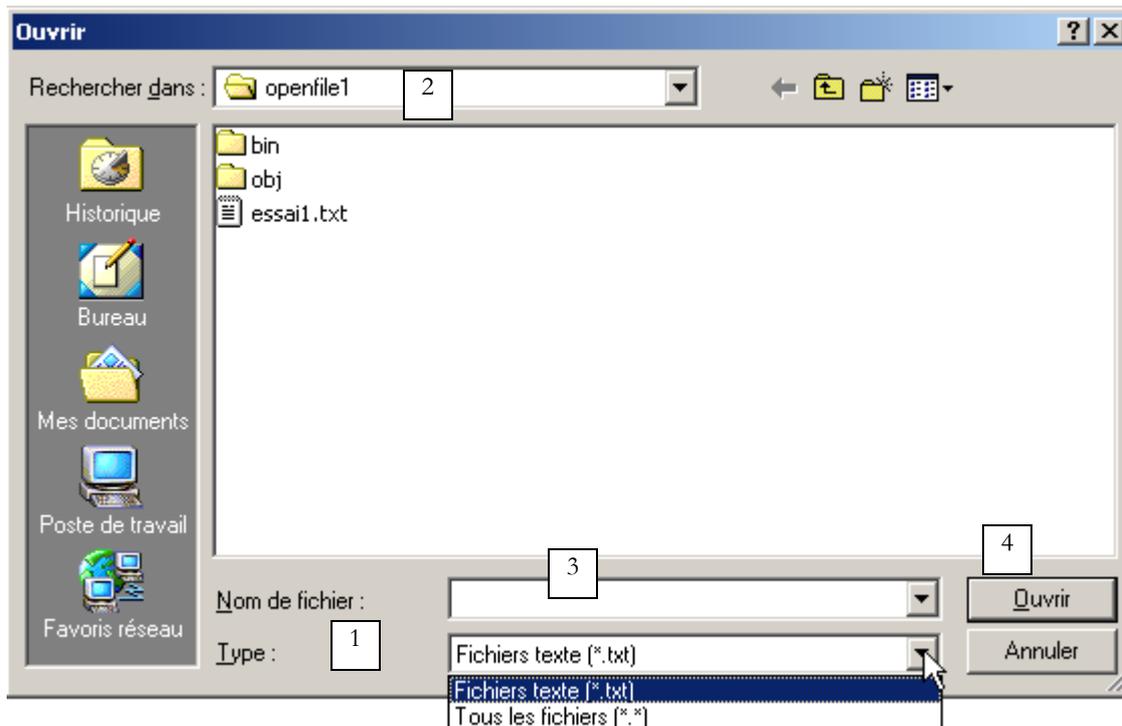
```
If saveFileDialog1.ShowDialog() = DialogResult.OK Then
```

- Pendant que la boîte de dialogue est affichée, l'utilisateur n'a plus accès au formulaire principal (boîte de dialogue dite modale). L'utilisateur fixe le nom du fichier à sauvegarder et quitte la boîte soit par le bouton Enregistrer, soit par le bouton Annuler soit en fermant la boîte. Le résultat de la méthode *ShowDialog* est *DialogResult.OK* uniquement si l'utilisateur a utilisé le bouton *Enregistrer* pour quitter la boîte de dialogue.
- Ceci fait, le nom du fichier à créer est maintenant dans la propriété *FileName* de l'objet *saveFileDialog1*. On est alors ramené à la création classique d'un fichier texte. On y écrit le contenu du *TextBox* : *txtTexte.Text* tout en gérant les exceptions qui peuvent se produire.

La classe **OpenFileDialog** est très proche de la classe *SaveFileDialog* et dérive de la même lignée de classes. De ces propriétés et méthodes nous retiendrons les suivantes :

string Filter	les types de fichiers proposés dans la liste déroulante des types de fichiers de la boîte de dialogue
int FilterIndex	le n° du type de fichier proposé par défaut dans la liste ci-dessus. Commence à 0.
string InitialDirectory	le dossier présenté initialement pour la recherche du fichier à ouvrir
string FileName	le nom du fichier à ouvrir indiqué par l'utilisateur
DialogResult.ShowDialog()	méthode qui affiche la boîte de dialogue de sauvegarde. Rend un résultat de type <i>DialogResult</i> .

La méthode *ShowDialog* affiche une boîte de dialogue analogue à la suivante :



- 1 liste déroulante construite à partir de la propriété **Filter**. Le type de fichier proposé par défaut est fixé par **FilterIndex**
- 2 dossier courant, fixé par **InitialDirectory** si cette propriété a été renseignée
- 3 nom du fichier choisi ou tapé directement par l'utilisateur. Sera disponible dans la propriété **FileName**
- 4 boutons **Ouvrir/Annuler**. Si le bouton *Ouvrir* est utilisé, la fonction *ShowDialog* rend le résultat **DialogResult.OK**

La procédure d'ouverture peut s'écrire ainsi :

```
Private Sub btnCharger_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnCharger.Click
    ' on charge un fichier texte dans la boîte de saisie
    ' on paramètre la boîte de dialogue openFileDialog1
    openFileDialog1.InitialDirectory = Application.ExecutablePath
    openFileDialog1.Filter = "Fichiers texte (*.txt)|*.txt|Tous les fichiers (*.*)|*.*"
    openFileDialog1.FilterIndex = 0
    ' on affiche la boîte de dialogue et on récupère son résultat
    If openFileDialog1.ShowDialog() = DialogResult.OK Then
        ' on récupère le nom du fichier
        Dim nomFichier As String = openFileDialog1.FileName
        Dim fichier As StreamReader = Nothing
        Try
            ' on ouvre le fichier en lecture
            fichier = New StreamReader(nomFichier)
            ' on lit tout le fichier et on le met dans le TextBox
            txtTexte.Text = fichier.ReadToEnd()
        Catch ex As Exception
            ' problème
            MessageBox.Show("Problème à la lecture du fichier (" + ex.Message + ")", "Erreur",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
            Return
        Finally
            ' on ferme le fichier
            Try
                fichier.Close()
            Catch
            End Try
        End Try
    End If
End Sub
```

- On fixe le dossier initial au dossier qui contient l'exécutable de l'application :

```
saveFileDialog1.InitialDirectory=Application.ExecutablePath
```

- On fixe les types de fichiers à présenter

```
saveFileDialog1.Filter = "Fichiers texte (*.txt)|*.txt|Tous les fichiers (*.*)|*.*"
```

- On fixe le type de fichier à présenter au début

```
saveFileDialog1.FilterIndex = 0
```

Ici, ce sont les fichiers de type \*.txt qui seront présentés tout d'abord à l'utilisateur.

- La boîte de dialogue est affichée et son résultat récupéré

```
If openFileDialog1.ShowDialog() = DialogResult.OK Then
```

Pendant que la boîte de dialogue est affichée, l'utilisateur n'a plus accès au formulaire principal (boîte de dialogue dite modale). L'utilisateur fixe le nom du fichier à ouvrir et quitte la boîte soit par le bouton Ouvrir, soit par le bouton Annuler soit en fermant la boîte. Le résultat de la méthode *ShowDialog* est *DialogResult.OK* uniquement si l'utilisateur a utilisé le bouton *Ouvrir* pour quitter la boîte de dialogue.

- Ceci fait, le nom du fichier à créer est maintenant dans la propriété *FileName* de l'objet *openFileDialog1*. On est alors ramené à la lecture classique d'un fichier texte. On notera la méthode qui permet de lire la totalité d'un fichier :

```
txtTexte.Text=fichier.ReadToEnd
```

- le contenu du fichier est mis dans le *TextBox txtTexte*. On gère les exceptions qui peuvent se produire.

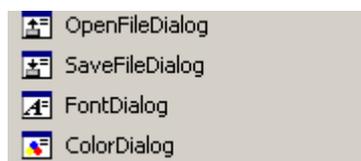
## 4.7.2 Boîtes de dialogue *FontColor* et *ColorDialog*

Nous continuons l'exemple précédent en présentant deux nouveaux boutons :



N°	type	nom	rôle
6	Button	btnCouleur	pour fixer la couleur des caractères du TextBox
7	Button	btnPolice	pour fixer la police de caractères du TextBox

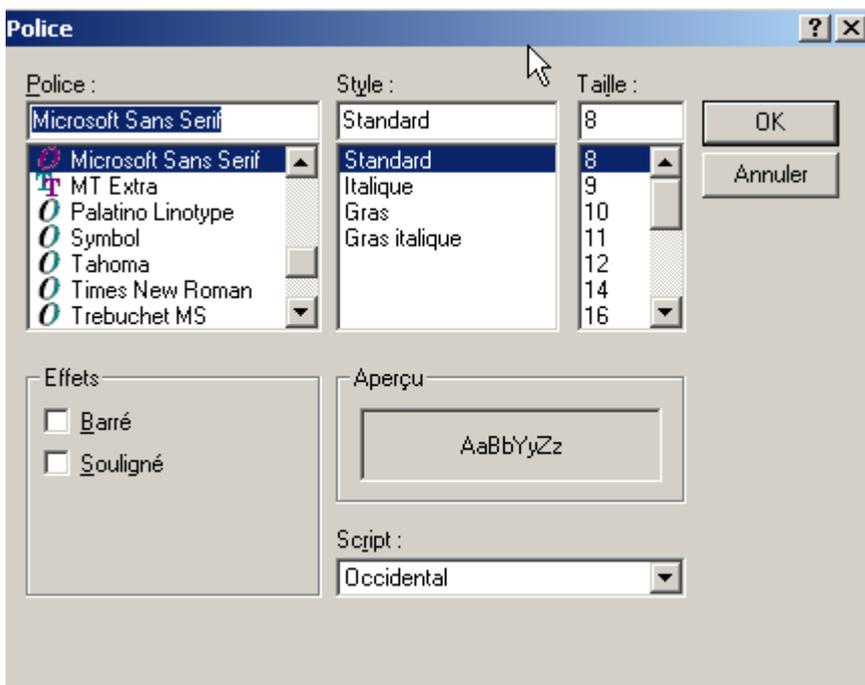
Nous déposons sur le formulaire un contrôle *ColorDialog* et un contrôle *FontDialog* :



Les classes *FontDialog* et *ColorDialog* ont une méthode *ShowDialog* analogue à la méthode *ShowDialog* des classes *OpenFileDialog* et *SaveFileDialog*. La méthode *ShowDialog* de la classe *ColorDialog* permet de choisir une couleur :



Si l'utilisateur quitte la boîte de dialogue avec le bouton *OK*, le résultat de la méthode *ShowDialog* est *DialogResult.OK* et la couleur choisie est dans la propriété *Color* de l'objet *ColorDialog* utilisé. La méthode *ShowDialog* de la classe *FontDialog* permet de choisir une police de caractères :



Si l'utilisateur quitte la boîte de dialogue avec le bouton *OK*, le résultat de la méthode *ShowDialog* est *DialogResult.OK* et la police choisie est dans la propriété *Font* de l'objet *FontDialog* utilisé. Nous avons les éléments pour traiter les clics sur les boutons *Couleur* et *Police* :

```
Private Sub btnCouleur_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnCouleur.Click
    ' choix d'une couleur de texte
    If colorDialog1.ShowDialog() = DialogResult.OK Then
        ' on change la propriété forecolor du TextBox
        txtTexte.ForeColor = colorDialog1.Color
    End If
End Sub

Private Sub btnPolice_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnPolice.Click
    ' choix d'une police de caractères
    If fontDialog1.ShowDialog() = DialogResult.OK Then
```

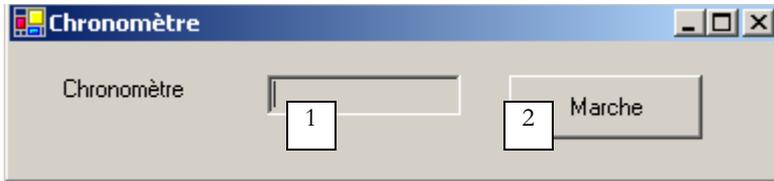
```

' on change la propriété font du TextBox
txtTexte.Font = fontDialog1.Font
End If
End Sub

```

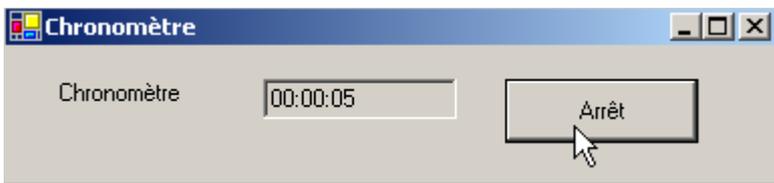
### 4.7.3 Timer

Nous nous proposons ici d'écrire l'application suivante :

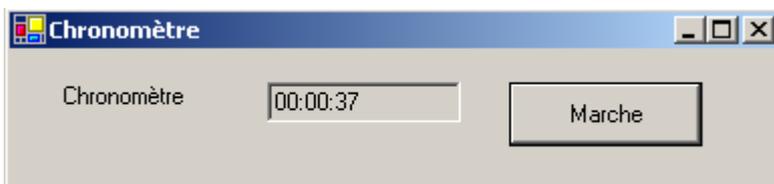


n°	Type	Nom	Rôle
1	TextBox, ReadOnly=true	txtChrono	affiche un chronomètre
2	Button	btnArretMarche	bouton Arrêt/Marche du chronomètre
3	Timer	timer1	composant émettant ici un événement toutes les secondes

Le chronomètre en marche :



Le chronomètre arrêté :



Pour changer toutes les secondes le contenu du TextBox *txtChrono*, il nous faut un composant qui génère un événement toutes les secondes, événement qu'on pourra intercepter pour mettre à jour l'affichage du chronomètre. Ce composant c'est le *Timer* :



Une fois ce composant installé sur le formulaire (dans la partie des composants non visuels), un objet de type *Timer* est créé dans le constructeur du formulaire. La classe *System.Windows.Forms.Timer* est définie comme suit :

**Timer, classe** [Visual Basic]

Génère des événements récurrents dans une application.

Pour obtenir une liste de tous les membres de ce type, consultez [Timer, membres](#).

[System.Object](#)

[System.MarshalByRefObject](#)

[System.ComponentModel.Component](#)

**System.Timers.Timer**

```
Public Class Timer
    Inherits Component
    Implements ISupportInitialize
```

De ses propriétés nous ne retiendrons que les suivantes :

Interval	nombre de millisecondes au bout duquel un événement <i>Tick</i> est émis.
Tick	l'événement produit à la fin de <i>Interval</i> millisecondes
Enabled	rend le timer actif (true) ou inactif (false)

Dans notre exemple le timer s'appelle *timer1* et *timer1.Interval* est mis à 1000 ms (1s). L'événement *Tick* se produira donc toutes les secondes. Le clic sur le bouton Arrêt/Marche est traité par la procédure suivante :

```
Private Sub btnArretMarche_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnArretMarche.Click
    ' arrêt ou marche ?
    If btnArretMarche.Text = "Marche" Then
        ' on note l'heure de début
        début = DateTime.Now
        ' on l'affiche
        txtChrono.Text = "00:00:00"
        ' on lance le timer
        timer1.Enabled = True
        ' on change le libellé du bouton
        btnArretMarche.Text = "Arrêt"
        ' fin
    Return
End If
    '
    If btnArretMarche.Text = "Arrêt" Then
        ' arrêt du timer
        timer1.Enabled = False
        ' on change le libellé du bouton
        btnArretMarche.Text = "Marche"
        ' fin
    Return
End If
End Sub

Private Sub timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles timer1.Tick
    ' une seconde s'est écoulée
    Dim maintenant As DateTime = DateTime.Now
    Dim durée As TimeSpan = DateTime.op_Subtraction(maintenant, début)
    txtChrono.Text = "" + durée.Hours.ToString("d2") + ":" + durée.Minutes.ToString("d2") + ":" +
durée.Seconds.ToString("d2")
End Sub
```

Le libellé du bouton Arrêt/Marche est soit "Arrêt" soit "Marche". On est donc obligé de faire un test sur ce libellé pour savoir quoi faire.

- dans le cas de "Marche", on note l'heure de début dans une variable qui est une variable globale de l'objet formulaire, le timer est lancé (Enabled=true) et le libellé du bouton passe à "Arrêt".
- dans le cas de "Arrêt", on arrête le timer (Enabled=false) et on passe le libellé du bouton à "Marche".

```
Public Class Timer1
    Inherits System.Windows.Forms.Form
    Private WithEvents timer1 As System.Windows.Forms.Timer
    Private WithEvents btnArretMarche As System.Windows.Forms.Button
    Private components As System.ComponentModel.IContainer
    Private WithEvents txtChrono As System.Windows.Forms.TextBox
    Private WithEvents label1 As System.Windows.Forms.Label

    ' variables d'instance
```

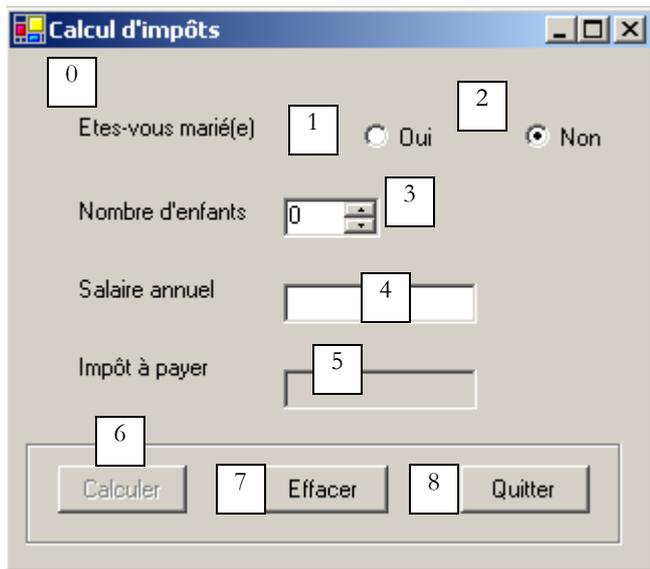
L'attribut *début* ci-dessus est connu dans toutes les méthodes de la classe. Il nous reste à traiter l'événement *Tick* sur l'objet *timer1*, événement qui se produit toutes les secondes :

```
Private Sub timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles timer1.Tick
' une seconde s'est écoulée
Dim maintenant As DateTime = DateTime.Now
Dim durée As TimeSpan = DateTime.op_Subtraction(maintenant, début)
txtChrono.Text = "" + durée.Hours.ToString("d2") + ":" + durée.Minutes.ToString("d2") + ":" +
durée.Seconds.ToString("d2")
End Sub
```

On calcule le temps écoulé depuis l'heure de lancement du chronomètre. On obtient un objet de type *TimeSpan* qui représente une durée dans le temps. Celle-ci doit être affichée dans le chronomètre sous la forme hh:mm:ss. Pour cela nous utilisons les propriétés *Hours*, *Minutes*, *Seconds* de l'objet *TimeSpan* qui représentent respectivement les heures, minutes, secondes de la durée que nous affichons au format *ToString("d2")* pour avoir un affichage sur 2 chiffres.

## 4.8 L'exemple IMPOTS

On reprend l'application IMPOTS déjà traitée deux fois. Nous y ajoutons maintenant une interface graphique :

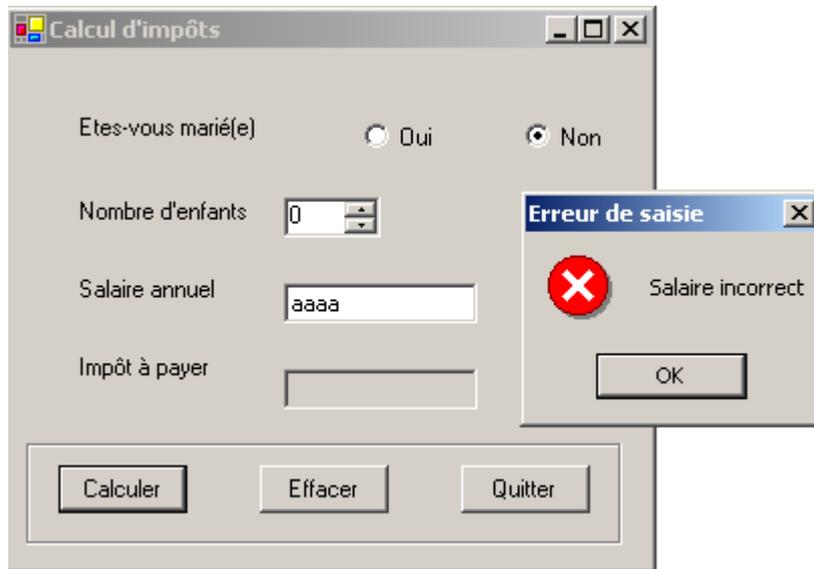


Les contrôles sont les suivants

n°	type	nom	rôle
1	RadioButton	rdOui	coché si marié
2	RadioButton	rdNon	coché si pas marié
3	NumericUpDown	incEnfants	nombre d'enfants du contribuable Minimum=0, Maximum=20, Increment=1
4	TextBox	txtSalaire	salaire annuel du contribuable en F
5	TextBox	txtImpots	montant de l'impôt à payer ReadOnly=true
6	Button	btnCalculer	lance le calcul de l'impôt
7	Button	btnEffacer	remet le formulaire dans son état initial lors du chargement
8	Button	btnQuitter	pour quitter l'application

### Règles de fonctionnement

- le bouton Calculer reste éteint tant qu'il n'y a rien dans le champ du salaire
- si lorsque le calcul est lancé, il s'avère que le salaire est incorrect, l'erreur est signalée :



Le programme est donné ci-dessous. Il utilise la classe *impot* créée dans le chapitre sur les classes. Une partie du code produit automatiquement pas VS.NET n'a pas été ici reproduit.

```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Drawing
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms
Imports System.Data

' classe formulaire
Public Class frmImpots
    Inherits System.Windows.Forms.Form
    Private WithEvents label1 As System.Windows.Forms.Label
    Private WithEvents rdOui As System.Windows.Forms.RadioButton
    Private WithEvents rdNon As System.Windows.Forms.RadioButton
    Private WithEvents label2 As System.Windows.Forms.Label
    Private WithEvents txtSalaire As System.Windows.Forms.TextBox
    Private WithEvents label3 As System.Windows.Forms.Label
    Private WithEvents label4 As System.Windows.Forms.Label
    Private WithEvents groupBox1 As System.Windows.Forms.GroupBox
    Private WithEvents btnCalculer As System.Windows.Forms.Button
    Private WithEvents btnEffacer As System.Windows.Forms.Button
    Private WithEvents btnQuitter As System.Windows.Forms.Button
    Private WithEvents txtImpots As System.Windows.Forms.TextBox
    Private components As System.ComponentModel.IContainer = Nothing
    Private WithEvents incEnfants As System.Windows.Forms.NumericUpDown

    ' tableaux de données nécessaires au calcul de l'impôt
    Private limites() As Decimal = {12620D, 13190D, 15640D, 24740D, 31810D, 39970D, 48360D, 55790D,
    92970D, 127860D, 151250D, 172040D, 195000D, 0D}
    Private coeffR() As Decimal = {0D, 0.05D, 0.1D, 0.15D, 0.2D, 0.25D, 0.3D, 0.35D, 0.4D, 0.45D, 0.55D,
    0.5D, 0.6D, 0.65D}
    Private coeffN() As Decimal = {0D, 631D, 1290.5D, 2072.5D, 3309.5D, 4900D, 6898.5D, 9316.5D, 12106D,
    16754.5D, 23147.5D, 30710D, 39312D, 49062D}
    ' objet impôt
    Private objImpôt As impot = Nothing

    Public Sub New()
        InitializeComponent()
        ' initialisation du formulaire
        btnEffacer_Click(Nothing, Nothing)
        btnCalculer.Enabled = False
        ' création d'un objet impôt

```

```

    Try
        objImpôt = New impot(limites, coeffR, coeffN)
    Catch ex As Exception
        MessageBox.Show("Impossible de créer l'objet impôt (" + ex.Message + ")", "Erreur",
        MessageBoxButtons.OK, MessageBoxIcon.Error)
        ' on inhibe le champ de saisie du salaire
        txtSalaire.Enabled = False
    End Try 'try-catch
End Sub

Protected Overloads Sub Dispose(ByVal disposing As Boolean)
....
End Sub

Private Sub InitializeComponent()
    Me.btnQuitter = New System.Windows.Forms.Button
    Me.groupBox1 = New System.Windows.Forms.GroupBox
    Me.btnEffacer = New System.Windows.Forms.Button
    Me.btnCalculer = New System.Windows.Forms.Button
    Me.txtSalaire = New System.Windows.Forms.TextBox
    Me.label1 = New System.Windows.Forms.Label
    Me.label2 = New System.Windows.Forms.Label
    Me.label3 = New System.Windows.Forms.Label
    Me.rdNon = New System.Windows.Forms.RadioButton
    Me.txtImpots = New System.Windows.Forms.TextBox
    Me.label4 = New System.Windows.Forms.Label
    Me.rdOui = New System.Windows.Forms.RadioButton
    Me.incEnfants = New System.Windows.Forms.NumericUpDown
    Me.groupBox1.SuspendLayout()
    CType(Me.incEnfants, System.ComponentModel.ISupportInitialize).BeginInit()
    Me.SuspendLayout()
....
End Sub 'InitializeComponent

Public Shared Sub Main()
    Application.Run(New frmImpots)
End Sub 'Main

Private Sub btnEffacer_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnEffacer.Click
    ' raz du formulaire
    incEnfants.Value = 0
    txtSalaire.Text = ""
    txtImpots.Text = ""
    rdNon.Checked = True
End Sub 'btnEffacer_Click

Private Sub txtSalaire_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles txtSalaire.TextChanged
    ' état du bouton Calculer
    btnCalculer.Enabled = txtSalaire.Text.Trim() <> ""
End Sub 'txtSalaire_TextChanged

Private Sub btnQuitter_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnQuitter.Click
    ' fin application
    Application.Exit()
End Sub 'btnQuitter_Click

Private Sub btnCalculer_Click(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles btnCalculer.Click
    ' le salaire est-il correct ?
    Dim intSalaire As Integer = 0
    Try
        ' récupération du salaire
        intSalaire = Integer.Parse(txtSalaire.Text)
        ' il doit être >=0
        If intSalaire < 0 Then
            Throw New Exception("")
        End If
    Catch ex As Exception
        ' msg d'erreur
        MessageBox.Show(Me, "Salaire incorrect", "Erreur de saisie", MessageBoxButtons.OK,
        MessageBoxIcon.Error)
        ' focus sur champ erroné
        txtSalaire.Focus()
        ' sélection du texte du champ de saisie

```

```

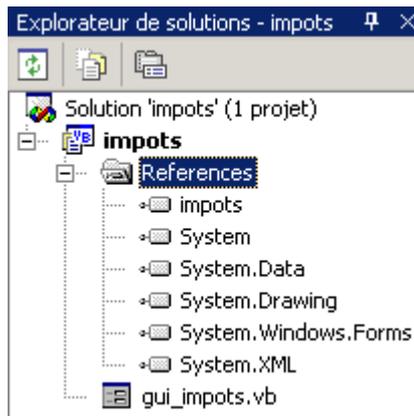
txtSalaire.SelectAll()
' retour à l'interface visuelle
Return
End Try 'try-catch
' le salaire est correct - on calcule l'impôt
txtImpots.Text = "" & CLng(objImpôt.calculer(rdOui.Checked, CInt(incEnfants.Value), intSalaire))
End Sub 'btnCalculer_Click
End Class

```

Nous utilisons ici l'assemblage **impots.dll** résultat de la compilation de la classe impots du chapitre 2. Rappelons que cet assemblage peut être produit en mode console par la commande

```
dos>vbc /t:library impots.vb
```

Cette commande produit le fichier *impots.dll* appelé assemblage. Cet assemblage peut être ensuite utilisé dans différents projets. Ici, dans notre projet sous VS.NET, nous utilisons la fenêtre des propriétés du projet :



Pour ajouter une référence (un assemblage) nous cliquons droit sur le lot clé References ci-dessus, nous prenons l'option [Ajouter une référence] et nous désignons l'assemblage [impots.dll] que nous avons pris soin de mettre dans le dossier du projet :

```

dos>dir
01/03/2004  14:39                9 250  gui_impots.vb
01/03/2004  14:37                4 096  impots.dll
01/03/2004  14:41               12 288  gui_impots.exe

```

Une fois inclus l'assemblage [impots.dll] dans le projet, la classe [impots] devient connue du projet. Auparavant elle ne l'est pas. Une autre méthode est d'inclure le source **impots.vb** dans le projet. Pour cela, dans la fenêtre des propriétés du projet, on clique droit sur le projet et on prend l'option [Ajouter/Ajouter un élément existant] et on désigne le fichier **impots.vb**.

# 5. Gestion d'événements

Nous avons dans le chapitre précédent abordé la notion d'événements liés à des composants. Nous voyons maintenant comment créer des événements dans nos propres classes. C'est une notion difficile qui n'intéressera pas les développeurs débutants auxquels il est conseillé de passer directement au chapitre suivant.

## 5.1 Objets delegate

L'instruction

```
Delegate Function opération(ByVal n1 As Integer, ByVal n2 As Integer) As Integer
```

définit un type appelé *opération* qui est en fait un prototype de fonction acceptant deux entiers et rendant un entier. C'est le mot clé **delegate** qui fait de *opération* une définition de prototype de fonction. Le type [opération] peut être affecté à une variable de la façon suivante :

```
Dim op As New opération(AddressOf [fonction])
```

Une variable de type [délégué] se construit comme un objet. Le paramètre du constructeur est la fonction associée à l'instance du délégué. En VB, cette fonction est donnée sous la forme `AddressOf fonction`, où [fonction] est le nom de la fonction. Une fois l'instance d'un délégué créée, elle peut être utilisée comme une fonction :

```
' on exécute le délégué  
Dim n As Integer = op(4, 7)
```

Le délégué [opération] a été défini comme recevant deux paramètres de type [integer] et renvoyant un résultat de même type. Donc son instance [op] ci-dessus est utilisée comme une telle fonction. Considérons l'exemple suivant :

```
' fonctions déléguées  
Imports System  
  
Public Class delegat1  
    ' définition d'un prototype de fonction  
    ' accepte 2 entiers en paramètre et rend un entier  
    Delegate Function opération(ByVal n1 As Integer, ByVal n2 As Integer) As Integer  
  
    ' deux méthodes d'instance correspondant au prototype  
    ' ajouter  
    Public Function ajouter(ByVal n1 As Integer, ByVal n2 As Integer) As Integer  
        Console.Out.WriteLine("ajouter(" & n1 & ", " & n2 & ")")  
        Return n1 + n2  
    End Function  
  
    ' soustraire  
    Public Function soustraire(ByVal n1 As Integer, ByVal n2 As Integer) As Integer  
        Console.Out.WriteLine("soustraire(" & n1 & ", " & n2 & ")")  
        Return n1 - n2  
    End Function  
  
    ' une méthode statique correspondant au prototype  
    Public Shared Function augmenter(ByVal n1 As Integer, ByVal n2 As Integer) As Integer  
        Console.Out.WriteLine("augmenter(" & n1 & ", " & n2 & ")")  
        Return n1 + 2 * n2  
    End Function  
  
    ' programme de test  
    Public Shared Sub Main()  
        ' on définit un objet de type opération pour y enregistrer des fonctions  
        ' on enregistre la fonction statique augmenter  
        Dim op As New opération(AddressOf delegat1.augmenter)  
        ' on exécute le délégué  
        Dim n As Integer = op(4, 7)  
        Console.Out.WriteLine("n=" & n)  
        ' création d'un objet c1 de type class1  
        Dim c1 As New delegat1  
        ' on enregistre dans le délégué la méthode ajouter de c1  
        op = New opération(AddressOf c1.ajouter)  
        ' exécution de l'objet délégué  
        n = op(2, 3)  
        Console.Out.WriteLine("n=" & n)  
        ' on enregistre dans le délégué la méthode soustraire de c1  
        op = New opération(AddressOf c1.soustraire)
```

```

    n = op(2, 3)
    Console.Out.WriteLine(("n=" & n))
End Sub
End Class

```

Les résultats de l'exécution sont les suivants :

```

augmenter(4,7)
n=18
ajouter(2,3)
n=5
soustraire(2,3)
n=-1

```

## 5.2 Gestion d'événements

A quoi peut servir un objet de type *delegate* ?

Comme nous le verrons dans l'exemple suivant, cela sert surtout à la gestion des événements. Une classe *C1* peut générer des événements *enti*. Lors d'un événement *enti*, un objet de type *C1* lancera l'exécution d'un objet *entiDéclenché* de type *delegate*. Toutes les fonctions enregistrées dans l'objet *delegate entiDéclenché* seront alors exécutées. Si un objet *C2* utilisant un objet *C1* veut être averti de l'occurrence de l'événement *enti* sur l'objet *C1*, il enregistrera l'une de ses méthodes *C2.f* dans l'objet délégué *C1.entiDéclenché* de l'objet *C1* afin que la méthode *C2.f* soit exécutée à chaque fois que l'événement *enti* se produit sur l'objet *C1*. Comme l'objet délégué *C1.entiDéclenché* peut enregistrer plusieurs fonctions, différents objets *Ci* pourront s'enregistrer auprès du délégué *C1.entiDéclenché* pour être prévenus de l'événement *enti* sur *C1*.

### 5.2.1 Déclaration d'un événement

Un événement se déclare de la façon suivante (1) :

```

Delegate Sub proc(liste d'arguments)
Public Event événement as proc

```

ou bien (2)

```

Public Event événement(liste d'arguments)

```

C'est le mot clé [Event] qui définit une donnée comme un événement. On associe à l'événement la signature que devront avoir les gestionnaires de celui-ci. On peut définir cette signature au moyen d'un délégué (méthode 1), soit directement (méthode 2).

### 5.2.2 Définir les gestionnaires d'un événement

Lorsqu'un événement est déclenché par l'instruction [RaiseEvent], il faut le traiter. Les procédures chargées de traiter les événements sont appelées des gestionnaires d'événements. A la création d'un objet [Event], seule la signature des gestionnaires de l'événement est déclarée. Cette signature est celle de l'objet [delegate] associé à l'événement. Ceci fait, on va pouvoir associer à l'événement des gestionnaires. Ceux-ci sont des procédures qui doivent avoir la même signature que le [delegate] associé à l'événement. Pour associer un gestionnaire d'événement à un événement, on utilise l'instruction [AddHandler] de syntaxe :

```

AddHandler event, AddressOf eventhandler

```

- **event** : variable événement à qui on associe un nouveau gestionnaire d'événements
- **eventhandler** : nom du gestionnaire d'événements - doit avoir la signature du [délégué] associé à l'événement

On peut associer autant de gestionnaires que l'on veut à un événement. Ils seront tous exécutés lorsque l'événement auquel ils sont associés sera déclenché.

### 5.2.3 Déclencher un événement

Pour déclencher un événement par programme, on utilise l'instruction [RaiseEvent] :

```

RaiseEvent eventname[( argumentlist )]

```

- **eventname** : variable événement (déclarée avec le mot clé Event)
- **argumentlist** : arguments du délégué associé à l'événement.

Tous les gestionnaires qui ont été associés à l'événement par l'instruction AddHandler vont être appelés. Ils vont recevoir comme paramètres, les paramètres définis dans la signature de l'événement.

## 5.2.4 Un exemple

Considérons l'exemple suivant :

```
'gestion d'événements
Imports System

Public Class myEventArgs
    Inherits EventArgs
    ' la classe d'un évt
    ' attribut
    Private _saisie As String

    ' constructeur
    Public Sub New(ByVal saisie As String)
        _saisie = saisie
    End Sub

    ' propriété saisie en lecture seule
    Public Overrides Function ToString() As String
        Return _saisie
    End Function
End Class

Public Class émetteur
    ' la classe émettrice d'un évt
    ' attribut
    Private _nom As String ' nom de l'émetteur

    ' constructeur
    Public Sub New(ByVal nom As String)
        _nom = nom
    End Sub

    ' ToString
    Public Overrides Function ToString() As String
        Return _nom
    End Function

    ' le prototype des fonctions chargées de traiter l'évt
    Delegate Sub _evtHandler(ByVal sender As Object, ByVal evt As myEventArgs)

    ' le pool des gestionnaires d'évts
    Public Event evtHandler As _evtHandler

    ' méthode de demande d'émission d'un évt
    Public Sub envoyerEvt(ByVal evt As myEventArgs)
        ' on prévient tous les abonnés
        ' on fait comme si l'évt provenait d'ici
        RaiseEvent evtHandler(Me, evt)
    End Sub
End Class

' une classe de traitement de l'évt
Public Class souscripteur
    ' attribut
    Private nom As String ' nom du souscripteur
    Private sender As émetteur ' l'émetteur des évts

    ' constructeur
    Public Sub New(ByVal nom As String, ByVal e As émetteur)
        ' on note le nom du souscripteur
        Me.nom = nom
        ' et l'émetteur des évts
        Me.sender = e
        ' on s'inscrit pour recevoir les evts de l'émetteur e
        AddHandler e(evtHandler, AddressOf traitementEvt)
    End Sub

    ' gestionnaire d'évt
    Public Sub traitementEvt(ByVal sender As Object, ByVal evt As myEventArgs)
        ' affichage evt
    End Sub
End Class
```

```

    Console.Out.WriteLine("L'objet [" + sender.ToString + "] a signalé la saisie erronée [" +
    evt.ToString + "] au souscripteur [" + nom + "]"))
End Sub
End Class

' un programme de test
Public Class test
    Public Shared Sub Main()
        ' création d'un émetteur d'evts
        Dim émetteur1 As New émetteur("émetteur1")
        ' création d'un tableau de souscripteurs
        ' pour les évts émis par émetteur1
        Dim souscripteurs() As souscripteur = {New souscripteur("s1", émetteur1), New souscripteur("s2",
émetteur1)}
        ' on lit une suite d'entiers au clavier
        ' dès que l'un est erroné, on émet un évt
        Console.Out.Write("Nombre entier (rien pour arrêter) : ")
        Dim saisie As String = Console.In.ReadLine().Trim()
        ' tant que la ligne saisie est non vide
        While saisie <> ""
            ' la saisie est-elle un nombre entier ?
            Try
                Dim n As Integer = Integer.Parse(saisie)
            Catch
                ' ce n'est pas un entier
                ' on prévient tout le monde
                émetteur1.envoyerEvt(New myEventArgs(saisie))
            End Try
            ' on prévient tout le monde
            ' nouvelle saisie
            Console.Out.Write("Nombre entier (rien pour arrêter) : ")
            saisie = Console.In.ReadLine().Trim()
        End While
        ' fin
        Environment.Exit(0)
    End Sub
End Class

```

Le code précédent est assez complexe. Détaillons-le. Dans une gestion d'événements, il y a un émetteur d'événements (*sender*) qui envoie le détail des événements (*EventArgs*) à des souscripteurs qui se sont déclarés intéressés par les événements en question. La classe émettrice des événements est ici la suivante :

```

Public Class émetteur
    ' la classe émettrice d'un évt
    ' attribut
    Private _nom As String ' nom de l'émetteur

    ' constructeur
    Public Sub New(ByVal nom As String)
        _nom = nom
    End Sub

    ' ToString
    Public Overrides Function ToString() As String
        Return _nom
    End Function

    ' le prototype des fonctions chargées de traiter l'évt
    Delegate Sub _evtHandler(ByVal sender As Object, ByVal evt As myEventArgs)

    ' le pool des gestionnaires d'évts
    Public Event evtHandler As _evtHandler

    ' méthode de demande d'émission d'un évt
    Public Sub envoyerEvt(ByVal evt As myEventArgs)
        ' on prévient tous les abonnés
        ' on fait comme si l'évt provenait d'ici
        RaiseEvent evtHandler(Me, evt)
    End Sub
End Class

```

Chaque objet *émetteur* a un *nom* fixé par construction. La méthode *ToString* a été redéfinie de telle façon que lorsqu'on transforme un objet *émetteur* en *string*, c'est son nom qu'on récupère. La classe définit un événement :

```

' le prototype des fonctions chargées de traiter l'évt
Delegate Sub _evtHandler(ByVal sender As Object, ByVal evt As myEventArgs)

' l'événement
Public Event evtHandler As _evtHandler

```

Le premier argument d'une fonction de traitement d'un événement sera l'objet qui émet l'événement. Le second argument sera de type *myEventArgs*, un objet qui donnera des détails sur l'événement et sur lequel nous reviendrons. Afin de pouvoir déclencher un événement de l'extérieur d'un objet *émetteur*, nous ajoutons à la classe la méthode *envoyerEvt* :

```
' méthode de demande d'émission d'un evt
Public Sub envoyerEvt(ByVal evt As myEventArgs)
    ' on prévient tous les abonnés
    ' on fait comme si l'evt provenait d'ici
    RaiseEvent evtHandler(Me, evt)
End Sub
```

Nous devons définir quel sera le type d'événements déclenchés par la classe *émetteur*. Nous avons pour cela défini la classe *myEventArgs* :

```
Public Class myEventArgs
    ' la classe d'un evt
    ' attribut
    Private _saisie As String

    ' constructeur
    Public Sub New(ByVal saisie As String)
        _saisie = saisie
    End Sub

    ' propriété saisie en lecture seule
    Public Overrides Function ToString() As String
        Return _saisie
    End Function
End Class
```

Les événements qui vont nous intéresser sont des saisies au clavier erronées. On va demander à un utilisateur de taper des nombres entiers au clavier et dès qu'il tapera une chaîne qui ne représente pas un entier, on déclenchera un événement. Comme détail de l'événement, nous nous contenterons de donner la saisie erronée. C'est le sens de l'attribut *\_saisie* de la classe. Un objet *myEventArgs* est donc construit avec pour paramètre la saisie erronée. On redéfinit par ailleurs la méthode *ToString* pour que lorsqu'on transforme un objet *myEventArgs* en chaîne, on obtienne l'attribut *\_saisie*.

Nous avons défini l'émetteur des événements et le type d'événements qu'il émet. Nous définissons ensuite le type des souscripteurs intéressés par ces événements.

```
' une classe de traitement de l'evt
Public Class souscripteur
    ' attribut
    Private nom As String ' nom du souscripteur
    Private sender As émetteur ' l'émetteur des évts

    ' constructeur
    Public Sub New(ByVal nom As String, ByVal e As émetteur)
        ' on note le nom du souscripteur
        Me.nom = nom
        ' et l'émetteur des évts
        Me.sender = e
        ' on s'inscrit pour recevoir les évts de l'émetteur e
        AddHandler e.evtHandler, AddressOf traitementEvt
    End Sub

    ' gestionnaire d'evt
    Public Sub traitementEvt(ByVal sender As Object, ByVal evt As myEventArgs)
        ' affichage evt
        Console.Out.WriteLine(("L'objet [" + sender.ToString + "] a signalé la saisie erronée [" +
            evt.ToString + "] au souscripteur [" + nom + "]"))
    End Sub
End Class
```

Un souscripteur sera défini par deux paramètres : son nom (attribut *nom*) et l'objet *émetteur* dont il veut traiter les événements (attribut *sender*). Ces deux paramètres seront passés au constructeur de l'objet. Au cours de cette même construction, le souscripteur s'abonne aux événements de l'émetteur :

```
' on s'inscrit pour recevoir les évts de l'émetteur e
AddHandler e.evtHandler, AddressOf traitementEvt
```

La fonction enregistrée auprès de l'émetteur est *traitementEvt*. Cette méthode de la classe *souscripteur* affiche les deux arguments qu'elle a reçus (*sender*, *evt*) ainsi que le nom du récepteur (*nom*). Ont été définis, le type des événements produits, le type de l'émetteur de ces événements, le type des souscripteurs. Il ne nous reste plus qu'à les mettre en oeuvre :

```
' un programme de test
Public Class test
```

```

Public Shared Sub Main()
    ' création d'un émetteur d'evts
    Dim émetteur1 As New émetteur("émetteur1")
    ' création d'un tableau de souscripteurs
    ' pour les évts émis par émetteur1
    Dim souscripteurs() As souscripteur = {New souscripteur("s1", émetteur1), New souscripteur("s2",
émetteur1)}
    ' on lit une suite d'entiers au clavier
    ' dès que l'un est erroné, on émet un évt
    Console.Out.Write("Nombre entier (rien pour arrêter) : ")
    Dim saisie As String = Console.In.ReadLine().Trim()
    ' tant que la ligne saisie est non vide
    While saisie <> ""
        ' la saisie est-elle un nombre entier ?
        Try
            Dim n As Integer = Integer.Parse(saisie)
        Catch
            ' ce n'est pas un entier
            ' on prévient tout le monde
            émetteur1.envoyerEvt(New myEventArgs(saisie))
        End Try
        ' on prévient tout le monde
        ' nouvelle saisie
        Console.Out.Write("Nombre entier (rien pour arrêter) : ")
        saisie = Console.In.ReadLine().Trim()
    End While
    ' fin
    Environment.Exit(0)
End Sub

```

Nous créons un objet *émetteur* :

```

' création d'un émetteur d'evts
Dim émetteur1 As New émetteur("émetteur1")

```

Nous créons un tableau de deux souscripteurs pour les événements émis par l'objet *émetteur1* :

```

' création d'un tableau de souscripteurs
' pour les évts émis par émetteur1
Dim souscripteurs() As souscripteur = {New souscripteur("s1", émetteur1), New souscripteur("s2",
émetteur1)}

```

Nous demandons à l'utilisateur de taper des nombres entiers au clavier. Dès qu'une saisie est erronée, nous demandons à *émetteur1* d'envoyer un événement à ses souscripteurs :

```

' on prévient tout le monde
émetteur1.envoyerEvt(New myEventArgs(saisie))

```

L'événement envoyé est de type *myEventArgs* et contient la saisie erronée. Les deux souscripteurs devraient recevoir cet événement et le signaler. C'est ce que montre l'exécution qui suit.

```

dos>evt1
Nombre entier (rien pour arrêter) : 4
Nombre entier (rien pour arrêter) : a
L'objet [émetteur1] a signalé la saisie erronée [a] au souscripteur [s1]
L'objet [émetteur1] a signalé la saisie erronée [a] au souscripteur [s2]
Nombre entier (rien pour arrêter) : 1.6
L'objet [émetteur1] a signalé la saisie erronée [1.6] au souscripteur [s1]
L'objet [émetteur1] a signalé la saisie erronée [1.6] au souscripteur [s2]
Nombre entier (rien pour arrêter) :

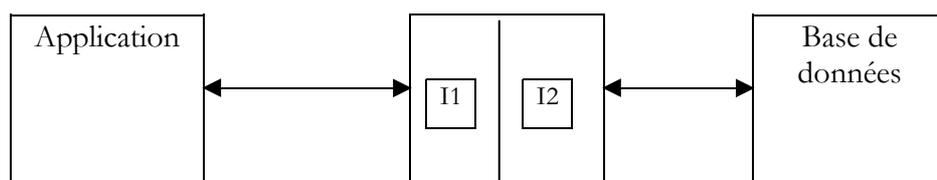
```

## 6. Accès aux bases de données

### 6.1 Généralités

Il existe de nombreuses bases de données pour les plate-formes windows. Pour y accéder, les applications passent au travers de programmes appelés **pilotes** (drivers).

Pilote de base de données



Dans le schéma ci-dessus, le pilote présente deux interfaces :

- l'interface I1 présentée à l'application
- l'interface I2 vers la base de données

Afin d'éviter qu'une application écrite pour une base de données B1 doive être ré-écrite si on migre vers une base de données B2 différente, un effort de normalisation a été fait sur l'interface I1. Si on utilise des bases de données utilisant des pilotes "normalisés", la base B1 sera fournie avec un pilote P1, la base B2 avec un pilote P2, et l'interface I1 de ces deux pilotes sera identique. Aussi n'aura-t-on pas à ré-écrire l'application. On pourra ainsi, par exemple, migrer une base de données ACCESS vers une base de données MySQL sans changer l'application.

Il existe deux types de pilotes normalisés :

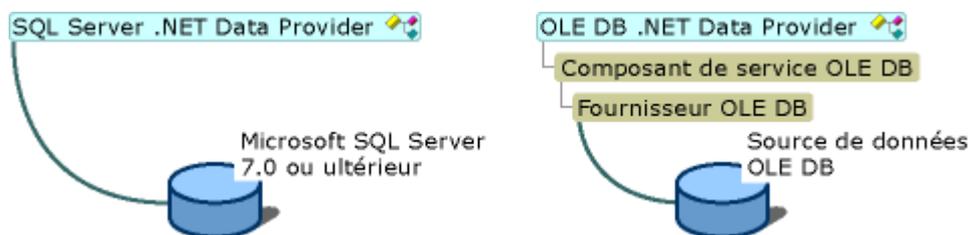
- les pilotes ODBC (Open DataBase Connectivity)
- les pilotes OLE DB (Object Linking and Embedding DataBase)

Les pilotes ODBC permettent l'accès à des bases de données. Les sources de données pour les pilotes OLE DB sont plus variées : bases de données, messageries, annuaires, ... Il n'y a pas de limite. Toute source de données peut faire l'objet d'un pilote Ole DB si un éditeur le décide. L'intérêt est évidemment grand : on a un accès uniforme à une grande variété de données.

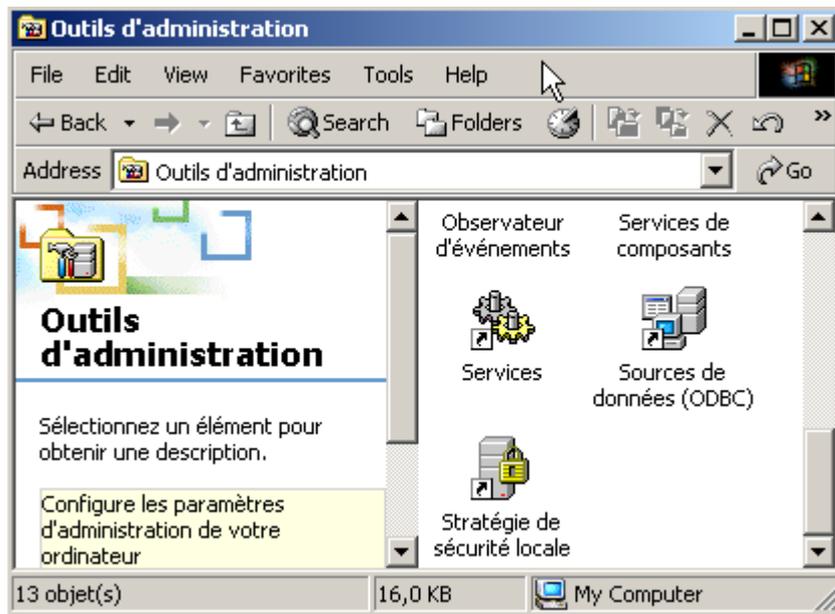
La plate-forme .NET est livrée avec deux types de classes d'accès aux données :

1. les classes SQL Server.NET
2. les classes Ole Db.NET

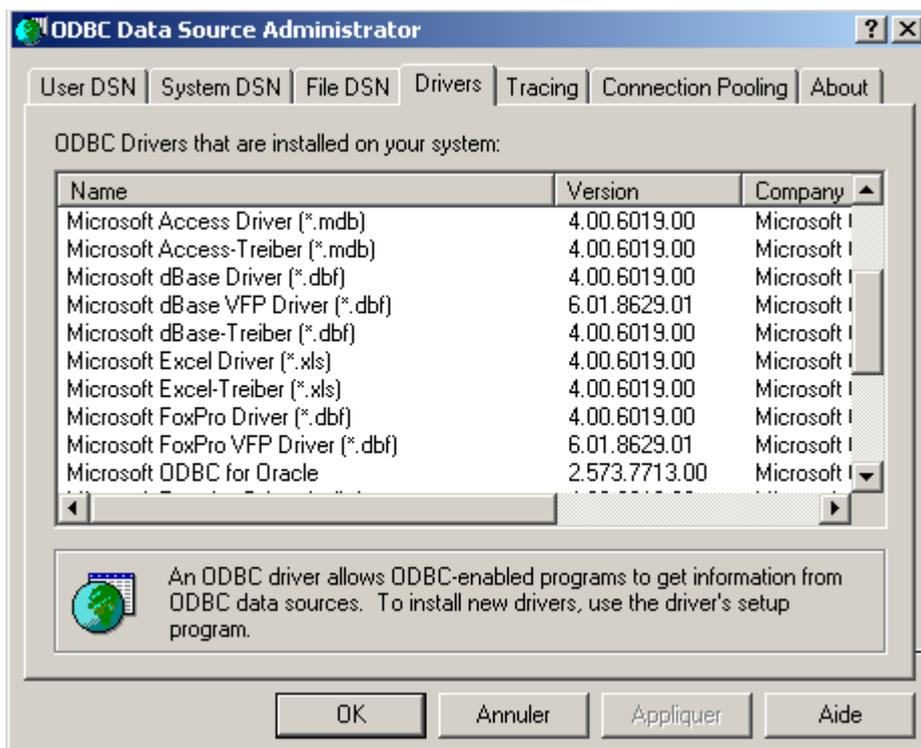
Les premières classes permettent un accès direct au SGBD SQL Server de Microsoft sans pilote intermédiaire. Les secondes permettent l'accès aux sources de données OLE DB.



La plate-forme .NET est fournie (mai 2002) avec trois pilotes OLE DB pour respectivement : SQL Server, Oracle et Microsoft Jet (Access). Si on veut travailler avec une base de données ayant un pilote ODBC mais pas de pilote OLE DB, on ne peut pas. Ainsi on ne peut pas travailler avec le SGBD MySQL qui (mai 2002) ne fournit pas de pilote OLE DB. Il existe cependant une série de classes permettant l'accès aux sources de données ODBC, les classes **odbc.net**. Elles ne sont pas livrées en standard avec le SDK et il faut aller les chercher sur le site de Microsoft. Dans les exemples qui vont suivre, nous utiliserons surtout ces classes ODBC car la plupart des bases de données sous windows sont livrées avec un tel pilote. Voici par exemple, une liste des pilotes ODBC installés sur une machine Win 2000 (*Menu Démarrer/Paramètres/Panneau de configuration/Outils d'administration*) :



On choisit l'icône *Source de données ODBC* :



## 6.2 Les deux modes d'exploitation d'une source de données

La plate-forme .NET permet l'exploitation d'une source de données de deux manières différentes :

1. mode connecté
2. mode déconnecté

En mode **connecté**, l'application

1. ouvre une connexion avec la source de données
2. travaille avec la source de données en lecture/écriture
3. ferme la connexion

En mode **déconnecté**, l'application

1. ouvre une connexion avec la source de données

Accès aux bases de données

2. obtient une copie mémoire de tout ou partie des données de la source
3. ferme la connexion
4. travaille avec la copie mémoire des données en lecture/écriture
5. lorsque le travail est fini, ouvre une connexion, envoie les données modifiées à la source de données pour qu'elle les prenne en compte, ferme la connexion

Dans les deux cas, c'est l'opération d'exploitation et de mise à jour des données qui prend du temps. Imaginons que ces mises à jour soient faites par un utilisateur faisant des saisies, cette opération peut prendre des dizaines de minutes. Pendant tout ce temps, en mode connecté, la connexion avec la base est maintenue et les modifications immédiatement répercutées. En mode déconnecté, il n'y a pas de connexion à la base pendant la mise à jour des données. Les modifications sont faites uniquement sur la copie mémoire. Elles sont répercutées sur la source de données en une seule fois lorsque tout est terminé.

Quels sont les avantages et inconvénients des deux méthodes ?

- Une connexion est coûteuse en ressources système. S'il y a beaucoup de connexions simultanées, le mode déconnecté permet de réduire leurs durées à un minimum. C'est le cas des applications web ayant des milliers d'utilisateurs.
- L'inconvénient du mode déconnecté est la gestion délicate des mises à jour simultanées. L'utilisateur U1 obtient des données au temps T1 et commence à les modifier. Au temps T2, l'utilisateur U2 accède lui aussi à la source de données et obtient les mêmes données. Entre-temps l'utilisateur U1 a modifié certaines données mais ne les a pas encore transmises à la source de données. U2 travaille donc avec des données dont certaines sont erronées. Les classes .NET offrent des solutions pour gérer ce problème mais il n'est pas simple à résoudre.
- En mode connecté, la mise à jour simultanée de données par plusieurs utilisateurs ne pose normalement pas de problème. La connexion avec la base de données étant maintenue, c'est la base de données elle-même qui gère ces mises à jour simultanées. Ainsi Oracle verrouille une ligne de la base de données dès qu'un utilisateur la modifie. Elle restera verrouillée donc inaccessible aux autres utilisateurs jusqu'à ce que celui qui l'a modifiée valide (commit) sa modification ou l'abandonne (rollback).
- Si les données doivent circuler sur le réseau, le mode déconnecté est à choisir. Il permet d'avoir une photo des données dans un objet appelé **dataset** qui représente une base de données à lui tout seul. Cet objet peut circuler sur le réseau entre machines.

Nous étudions d'abord le mode connecté.

## 6.3 Accès aux données en mode connecté

### 6.3.1 Les bases de données de l'exemple

Nous considérons une base de données ACCESS appelée *articles.mdb* et n'ayant qu'une table appelée ARTICLES avec la structure suivante :

nom	type
code	code de l'article sur 4 caractères
nom	son nom (chaîne de caractères)
prix	son prix (réel)
stock_actuel	son stock actuel (entier)
stock_minimum	le stock minimum (entier) en-deça duquel il faut réapprovisionner l'article

Son contenu de départ est le suivant :

articles : Table					
	code	nom	prix	stock_actuel	stock_minimum
▶	a300	vélo	2 500,00 F	10	5
	b300	pompe	56,00 F	62	45
	c300	arc	3 500,00 F	10	20
	d300	flèches - lot de	780,00 F	12	20
	e300	combinaison de	2 800,00 F	34	7
	f300	bouteilles d'oxy	800,00 F	10	5

Nous utiliserons cette base aussi bien au travers d'un pilote ODBC qu'un pilote OLE DB afin de montrer la similitude des deux approches et parce que nous disposons de ces deux types de pilotes pour ACCESS.

Nous utiliserons également une base MySQL DBARTICLES ayant la même unique table ARTICLES, le même contenu et accédé au travers d'un pilote ODBC, afin de montrer que l'application écrite pour exploiter la base ACCESS n'a pas à être modifiée pour utiliser la base MySQL. La base DBARTICLES est accessible à un utilisateur appelé *admarticles* avec le mot de passe *mdparticles*. La copie d'écran suivante montre le contenu de la base MySQL :

```
C:\mysql\bin>mysql --database=dbarticles --user=admarticles --password=mdparticles
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 3.23.49-max-debug

Type 'help' for help.

mysql> show tables;
+-----+
| Tables_in_dbarticles |
+-----+
| articles              |
+-----+
1 row in set (0.01 sec)

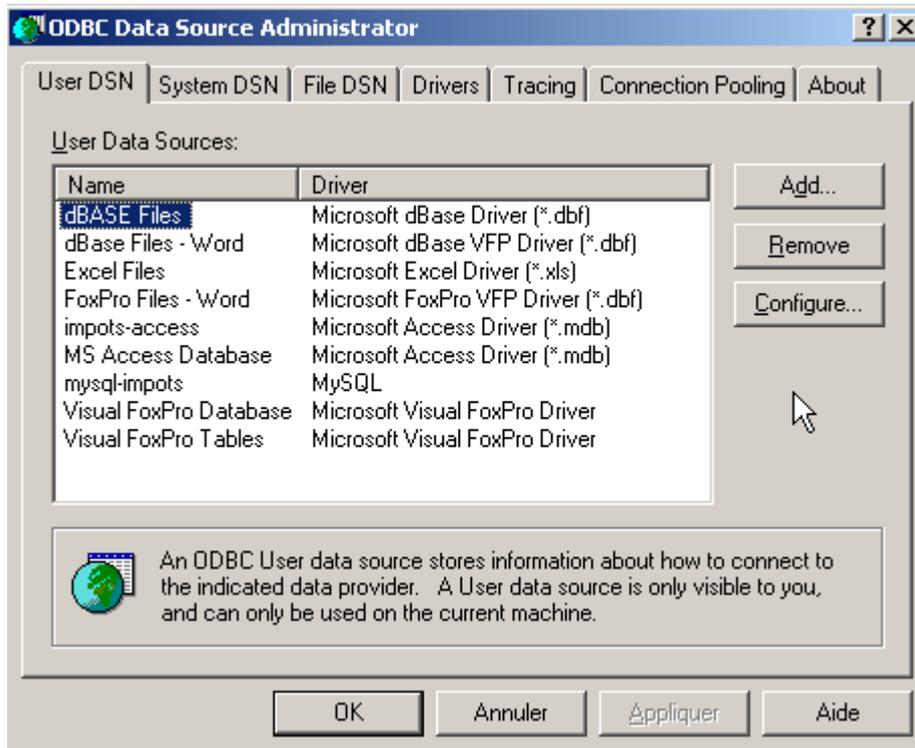
mysql> select * from articles;
+-----+-----+-----+-----+-----+
| code | nom                | prix | stock_actuel | stock_minimum |
+-----+-----+-----+-----+-----+
| a300 | vlo              | 2500 | 10           | 5             |
| b300 | pompe              | 56   | 62           | 45            |
| c300 | arc                 | 3500 | 10           | 20            |
| d300 | flches - lot de 6 | 780  | 12           | 20            |
| e300 | combinaison de plonge | 2800 | 34           | 7             |
| f300 | bouteilles d'oxygne | 800  | 10           | 5             |
+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)

mysql> describe articles;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| code           | text          | YES  |     | NULL    |       |
| nom            | text          | YES  |     | NULL    |       |
| prix           | double        | YES  |     | NULL    |       |
| stock_actuel   | smallint(6)  | YES  |     | NULL    |       |
| stock_minimum | smallint(6)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

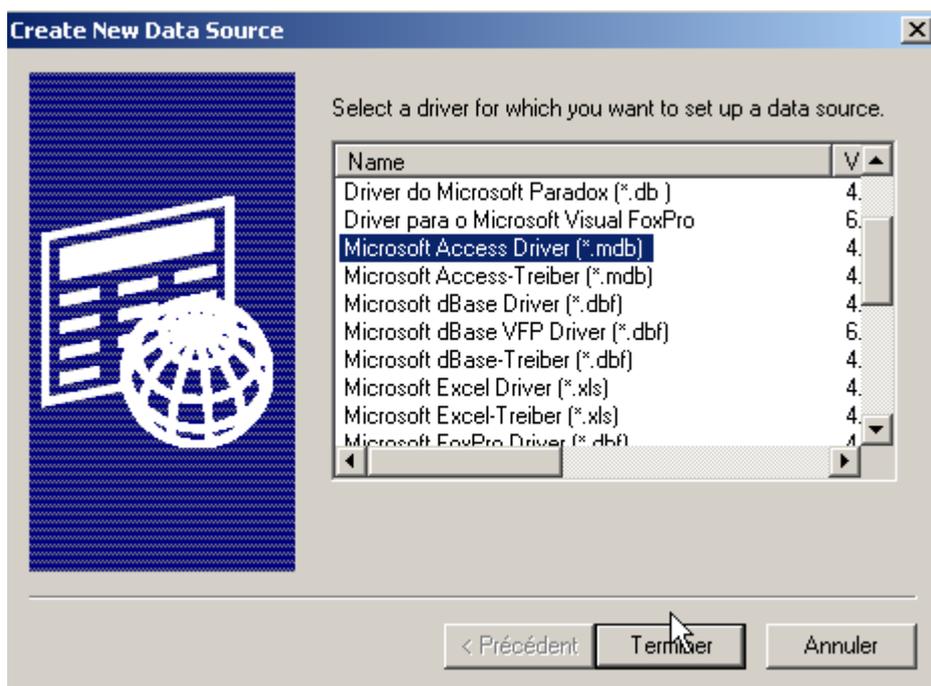
mysql> exit
Bye
```

Pour dfinir la base ACCESS comme source de donnes ODBC, procdez comme suit :

- activez l'administrateur de sources de donnes ODBC comme il a t montr plus haut et slectionnez l'onglet User DSN (DSN=Data Source Name)



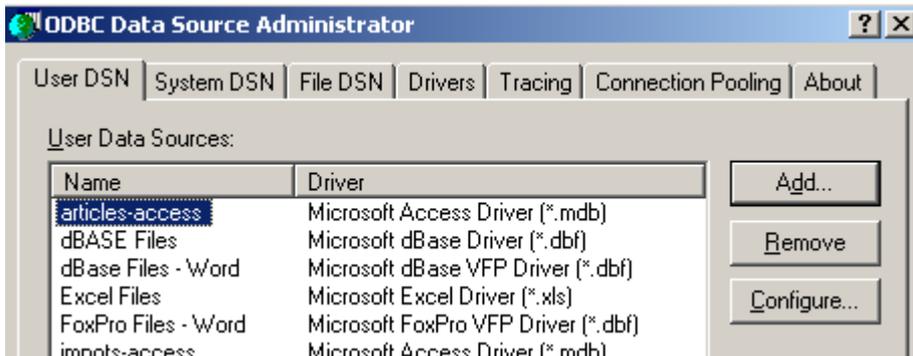
- ajoutez une source avec le bouton *Add*, indiquez que cette source est accessible via un pilote *Access* et faites *Terminer* :



- Donnez le nom *articles-access* à la source de données, mettez une description libre et utilisez le bouton *Sélectionner* pour désigner le fichier *.mdb* de la base. Terminez par *OK*.

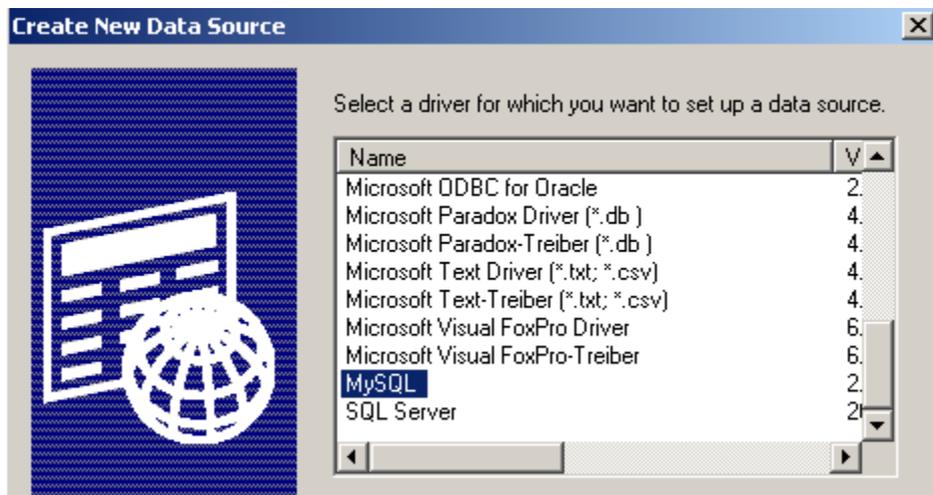


La nouvelle source de données apparaît alors dans la liste des sources DSN utilisateur :

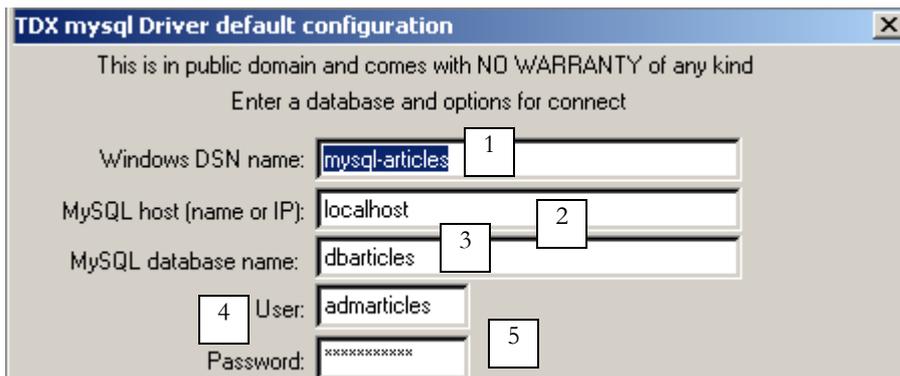


Pour définir la base MySQL DBARTICLES comme source de données ODBC, procédez comme suit :

- activez l'administrateur de sources de données ODBC comme il a été montré plus haut et sélectionnez l'onglet *User DSN*. Ajoutez une nouvelle source de données avec *Add* et sélectionnez le pilote ODBC de MySQL.



- Faites *Terminer*. Apparaît alors une page de configuration de la source MySQL :



- dans (1) on donne un nom à notre source de données ODBC
- dans (2) on indique la machine sur laquelle se trouve le serveur MySQL. Ici nous mettons *localhost* pour indiquer qu'il est sur la même machine que notre application. Si le serveur MySQL était sur une machine M distante, on mettrait là son nom et notre application fonctionnerait alors avec une base de données distante sans modification.
- dans (3) on met le nom de la base. Ici elle s'appelle DBARTICLES.
- dans (4) on met le login *admarticles* et dans (5) le mot de passe *mdparticles*.

## 6.3.2 Utilisation d'un pilote ODBC

Dans une application utilisant une base de données en mode connecté, on trouvera généralement les étapes suivantes :

1. Connexion à la base de données
2. Émissions de requêtes SQL vers la base
3. Réception et traitement des résultats de ces requêtes
4. Fermeture de la connexion

Les étapes 2 et 3 sont réalisées de façon répétée, la fermeture de connexion n'ayant lieu qu'à la fin de l'exploitation de la base. C'est un schéma relativement classique dont vous avez peut-être l'habitude si vous avez exploité une base de données de façon interactive. Ces étapes sont les mêmes que la base soit utilisée au travers d'un pilote ODBC ou d'un pilote OLE DB. Nous présentons ci-dessous un exemple avec les classes .NET de gestion des sources de données ODBC. Le programme s'appelle *liste* et admet comme paramètre le nom DSN d'une source de données ODBC ayant une table ARTICLES. Il affiche alors le contenu de cette table :

```
dos>liste
syntaxe : pg dsnArticles

dos>liste articles-access

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                2500 10 5
b300 pompe               56 62 45
c300 arc                 3500 10 20
d300 flèches - lot de 6  780 12 20
e300 combinaison de plongée 2800 34 7
f300 bouteilles d'oxygène 800 10 5

dos>liste mysql-artices
Erreur d'exploitation de la base de données (ERROR [IM002] [Microsoft][ODBC Driver Manager] Data source
name not found and no default driver specified)

dos>liste mysql-articles

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                2500 10 5
b300 pompe               56 62 45
c300 arc                 3500 10 20
d300 flèches - lot de 6  780 12 20
e300 combinaison de plongée 2800 34 7
f300 bouteilles d'oxygène 800 10 5
```

Sur les résultats ci-dessus, nous voyons que le programme a listé aussi bien le contenu de la base ACCESS que de la base MySQL. Etudions maintenant le code de ce programme :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.Data.Odbc
Imports Microsoft.VisualBasic

Module db1
  Sub main(ByVal args As String())
    ' application console
    ' affiche le contenu d'une table ARTICLES d'une base DSN
```

```

' dont le nom est passé en paramètre
Const syntaxe As String = "syntaxe : pg dsnArticles"
Const tabArticles As String = "articles" ' la table des articles

' vérification des paramètres
' a-t-on 1 paramètre
If args.Length <> 1 Then
    ' msg d'erreur
    Console.Error.WriteLine(syntaxe)
    ' fin
    Environment.Exit(1)
End If

' on récupère le paramètre
Dim dsnArticles As String = args(0) ' la base DSN
' préparation de la connexion à la bd
Dim articlesConn As OdbcConnection = Nothing ' la connexion
Dim myReader As OdbcDataReader = Nothing ' le lecteur de données

' on tente d'accéder à la base de données
Try
    ' chaîne de connexion à la base
    Dim connectString As String = "DSN=" + dsnArticles + ";"
    articlesConn = New OdbcConnection(connectString)
    articlesConn.Open()

    ' exécution d'une commande SQL
    Dim sqlText As String = "select * from " + tabArticles
    Dim myOdbcCommand As New OdbcCommand(sqlText)
    myOdbcCommand.Connection = articlesConn
    myReader = myOdbcCommand.ExecuteReader()

    ' Exploitation de la table récupérée
    ' affichage des colonnes
    Dim ligne As String = ""
    Dim i As Integer
    For i = 0 To (myReader.FieldCount - 1) - 1
        ligne += myReader.GetName(i) + ","
    Next i
    ligne += myReader.GetName(i)
    Console.Out.WriteLine((ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf + ligne +
ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf))

    ' affichage des données
    While myReader.Read()
        ' exploitation ligne courante
        ligne = ""
        For i = 0 To myReader.FieldCount - 1
            ligne += myReader(i).ToString + " "
        Next i
        Console.WriteLine(ligne)
    End While
Catch ex As Exception
    Console.Error.WriteLine(("Erreur d'exploitation de la base de données " + ex.Message + "))
    Environment.Exit(2)
Finally
    ' fermeture lecteur
    myReader.Close()
    ' fermeture connexion
    articlesConn.Close()
End Try
End Sub
End Module

```

Les classes de gestion des sources ODBC se trouvent dans l'espace de noms *Microsoft.Data.Odbc* qu'on doit donc importer. Par ailleurs, un certain nombre de classes se trouve dans l'espace de noms *System.Data*.

```
Imports System.Data
Imports Microsoft.Data.Odbc
```

Les espaces de noms utilisés par le programme sont dans différents assemblages. On compile le programme avec la commande suivante :

```
dos>vbc /r:microsoft.data.odbc.dll /r:microsoft.visualbasic.dll /r:system.dll /r:system.data.dll db1.vb
```

### 6.3.2.1 La phase de connexion

Une connexion ODBC utilise la classe *OdbcConnection*. Le constructeur de cette classe admet comme paramètre ce qu'on appelle une **chaîne de connexion**. Celle-ci est une chaîne de caractères qui définit tous les paramètres nécessaires pour que la connexion à la base de données puisse se faire. Ces paramètres peuvent être très nombreux et donc la chaîne complexe. La chaîne a la forme "*param1=valeur1;param2=valeur2;...;paramj=valeurj*";". Voici quelques paramètres *paramj* possibles :

uid	nom d'un utilisateur qui va accéder à la base de données
password	mot de passe de cet utilisateur
dsn	nom DSN de la base si elle en a un
data source	nom de la base de données accédée
...	

Si on définit une source de données comme source de données ODBC à l'aide de l'administrateur de sources de données ODBC, ces paramètres ont déjà été donnés et enregistrés. Il suffit alors de passer le paramètre DSN qui donne le nom DSN de la source de données. C'est ce qui est fait ici :

```
' préparation de la connexion à la bd
Dim articlesConn As OdbcConnection = Nothing ' la connexion
Dim myReader As OdbcDataReader = Nothing ' le lecteur de données
Try
    ' on tente d'accéder à la base de données
    ' chaîne de connexion à la base
    Dim connectString As String = "DSN=" + dsnArticles + ";"
    articlesConn = New OdbcConnection(connectString)
    articlesConn.Open()
```

Une fois l'objet *OdbcConnection* construit, on ouvre la connexion avec la méthode *Open*. Cette ouverture peut échouer comme toute autre opération sur la base. C'est pourquoi l'ensemble du code d'accès à la base est-il dans un *try-catch*. Une fois la connexion établie, on peut émettre des requêtes SQL sur la base.

### 6.3.2.2 Émettre des requêtes SQL

Pour émettre des requêtes SQL, il nous faut un objet *Command*, ici plus exactement un objet *OdbcCommand* puisque nous utilisons une source de données ODBC. La classe *OdbcCommand* a plusieurs constructeurs :

- *OdbcCommand()* : crée un objet *Command* vide. Il faudra pour l'utiliser préciser ultérieurement diverses propriétés :
  - **CommandText** : le texte de la requête SQL à exécuter
  - **Connection** : l'objet *OdbcConnection* représentant la connexion à la base de données sur laquelle la requête sera faite
  - **CommandType** : le type de la requête SQL. Il y a trois valeurs possibles
    1. *CommandType.Text* : la propriété *CommandText* contient le texte d'une requête SQL (valeur par défaut)
    2. *CommandType.StoredProcedure* : la propriété *CommandText* contient le nom d'une procédure stockée dans la base
    3. *CommandType.TableDirect* : la propriété *CommandText* contient le nom d'une table T. Equivalent à *select \* from T*. N'existe que pour les pilotes OLE DB.
- *OdbcCommand(string sqlText)* : le paramètre *sqlText* sera affecté à la propriété *CommandText*. C'est le texte de la requête SQL à exécuter. La connexion devra être précisée dans la propriété *Connection*.
- *OdbcCommand(string sqlText, OdbcConnection connexion)* : le paramètre *sqlText* sera affecté à la propriété *CommandText* et le paramètre *connexion* à la propriété *Connection*.

Pour émettre la requête SQL, on dispose de deux méthodes :

- **OdbcDataReader ExecuteReader()** : envoie la requête SELECT de *CommandText* à la connexion *Connection* et construit un objet *OdbcDataReader* permettant l'accès à toutes les lignes de la table résultat du *select*
- **int ExecuteNonQuery()** : envoie la requête de mise à jour (INSERT, UPDATE, DELETE) de *CommandText* à la connexion *Connection* et rend le nombre de lignes affectées par cette mise à jour.

Dans notre exemple, après avoir ouvert la connexion à la base, nous émettons une requête SQL SELECT pour avoir le contenu de la table ARTICLES :

```
' exécution d'une commande SQL
Dim sqlText As String = "select * from " + tabArticles
Dim myOdbcCommand As New OdbcCommand(sqlText)
myOdbcCommand.Connection = articlesConn
```

```
myReader = myOdbcCommand.ExecuteReader()
```

Une requête d'interrogation est classiquement une requête du type :

```
select col1, col2,... from table1, table2,...
where condition
order by expression
...
```

Seuls les mots clés de la première ligne sont obligatoires, les autres sont facultatifs. Il existe d'autres mots clés non présentés ici.

1. Une jointure est faite avec toutes les tables qui sont derrière le mot clé **from**
2. Seules les colonnes qui sont derrière le mot clé **select** sont conservées
3. Seules les lignes vérifiant la condition du mot clé **where** sont conservées
4. Les lignes résultantes ordonnées selon l'expression du mot clé **order by** forment le résultat de la requête.

Le résultat d'un *select* est une table. Si on considère la table ARTICLES précédente et qu'on veuille les noms des articles dont le stock actuel est au-dessous du seuil minimal, on écrira :

```
select nom from articles where stock_actuel < stock_minimum
```

Si on les veut par ordre alphabétique des noms, on écrira :

```
select nom from articles where stock_actuel < stock_minimum order by nom
```

### 6.3.2.3 Exploitation du résultat d'une requête SELECT

Le résultat d'une requête SELECT en mode non connecté est un objet *DataReader*, ici un objet *OdbcDataReader*. Cet objet permet d'obtenir séquentiellement toutes les lignes du résultat et d'avoir des informations sur les colonnes de ces résultats. Examinons quelques propriétés et méthodes de cette classe :

FieldCount	le nombre de colonnes de la table
Item	<i>Item(i)</i> représente la colonne n° i de la ligne courante du résultat
XXX GetXXX(i)	la valeur de la colonne n° i de la ligne courante rendue comme type XXX(Int16, Int32, Int64, Double, String, Boolean, ...)
string GetName(i)	nom de la colonne n° i
Close()	ferme l'objet <i>OdbcDataReader</i> et libère les ressources associées
bool Read()	avance d'une ligne dans la table des résultats. Rend faux si cela n'est pas possible. La nouvelle ligne devient la ligne courante du lecteur.

L'exploitation du résultat d'un *select* est typiquement une exploitation séquentielle analogue à celle des fichiers texte : on ne peut qu'avancer dans la table, pas reculer :

```
While myReader.Read()
' on a une ligne - on l'exploite
...
' ligne suivante
end while
```

Ces explications suffisent à comprendre le code suivant de notre exemple :

```
' Exploitation de la table récupérée
' affichage des colonnes
Dim ligne As String = ""
Dim i As Integer
For i = 0 To (myReader.FieldCount - 1) - 1
    ligne += myReader.GetName(i) + ","
Next i
ligne += myReader.GetName(i)
Console.Out.WriteLine((ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf + ligne +
ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf))

' affichage des données
While myReader.Read()
' exploitation ligne courante
ligne = ""
For i = 0 To myReader.FieldCount - 1
```

```

    ligne += myReader(i).ToString + " "
Next i
Console.WriteLine(ligne)
End While

```

La seule difficulté est dans l'instruction où les valeurs des différentes colonnes de la ligne courante sont concaténées :

```

For i = 0 To myReader.FieldCount - 1
    ligne += myReader(i).ToString + " "
Next i

```

La notation `ligne+=myReader(i).ToString` est traduit par `ligne+=myReader.Item(i).ToString()` où `Item(i)` est la valeur de la colonne `i` de la ligne courante.

### 6.3.2.4 Libération des ressources

Les classes `OdbcReader` et `OdbcConnection` possèdent toutes deux une méthode **Close()** qui libère les ressources associées aux objets ainsi fermés.

```

' fermeture lecteur
myReader.Close()
' fermeture connexion
articlesConn.Close()

```

## 6.3.3 Utilisation d'un pilote OLE DB

Nous reprenons le même exemple, cette fois avec une base accédée via un pilote OLE DB. La plate-forme .NET fournit un tel pilote pour les bases ACCESS. Aussi allons-nous utiliser la même base `articles.mdb` que précédemment. Nous cherchons à montrer ici que si les classes changent, les concepts restent les mêmes :

- la connexion est représentée par un objet **OleDbConnection**
- une requête SQL est émise grâce à un objet **OleDbCommand**
- si cette requête est une clause SELECT, on obtiendra en retour un objet **OleDbDataReader** pour accéder aux lignes de la table résultat

Ces classes sont dans l'espace de noms `System.Data.OleDb`. Le programme précédent peut être transformé aisément pour gérer une base OLE DB :

- on remplace partout `OdbcXX` par `OleDbXX`
- on modifie la chaîne de connexion. Pour une base ACCESS sans login/mot de passe, la chaîne de connexion est `Provider=Microsoft.JET.OLEDB.4.0;Data Source=[fichier.mdb]`. La partie paramétrable de cette chaîne est le nom du fichier ACCESS à utiliser. Nous modifierons notre programme pour qu'il accepte en paramètre le nom de ce fichier.
- l'espace de noms à importer est maintenant `System.Data.OleDb`.

Notre programme devient le suivant :

```

' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.Data.Odbc
Imports Microsoft.VisualBasic
Imports System.Data.OleDb

Module db2
    Public Sub Main(ByVal args() As String)

        ' application console
        ' affiche le contenu d'une table ARTICLES d'une base DSN
        ' dont le nom est passé en paramètre
        Const syntaxe As String = "syntaxe : pg base_access_articles"
        Const tabArticles As String = "articles" ' la table des articles

        ' vérification des paramètres
        ' a-t-on 1 paramètre
        If args.Length <> 1 Then
            ' msg d'erreur
            Console.Error.WriteLine(syntaxe)
        ' fin
    End Sub
End Module

```

```

Environment.Exit(1)
End If

' on récupère le paramètre
Dim dbArticles As String = args(0) ' la base de données

' préparation de la connexion à la bd
Dim articlesConn As OleDbConnection = Nothing ' la connexion
Dim myReader As OleDbDataReader = Nothing ' le lecteur de données

' on tente d'accéder à la base de données

Try
' chaîne de connexion à la base
Dim connectString As String = "Provider=Microsoft.JET.OLEDB.4.0;Data Source=" + dbArticles + ";"
articlesConn = New OleDbConnection(connectString)
articlesConn.Open()

' exécution d'une commande SQL
Dim sqlText As String = "select * from " + tabArticles
Dim myOleDbCommand As New OleDbCommand(sqlText)
myOleDbCommand.Connection = articlesConn
myReader = myOleDbCommand.ExecuteReader()

' Exploitation de la table récupérée
' affichage des colonnes
Dim ligne As String = ""
Dim i As Integer
For i = 0 To (myReader.FieldCount - 1) - 1
    ligne += myReader.GetName(i) + ","
Next i
ligne += myReader.GetName(i)
Console.Out.WriteLine((ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf + ligne +
ControlChars.Lf + "".PadLeft(ligne.Length, "-") + ControlChars.Lf))
' affichage des données
While myReader.Read()
' exploitation ligne courante
ligne = ""
For i = 0 To myReader.FieldCount - 1
    ligne += myReader(i).ToString + " "
Next i
Console.WriteLine(ligne)
End While
Catch ex As Exception
Console.Error.WriteLine("Erreur d'exploitation de la base de données (" + ex.Message + ")")
Environment.Exit(2)
Finally
' fermeture lecteur
myReader.Close()
' fermeture connexion
articlesConn.Close()
End Try
' fin
Environment.Exit(0)
End Sub
End Module

```

Les résultats obtenus :

```

dos>vbc liste.vb

E:\data\serge\MSNET\vb.net\adonet\6>dir
07/05/2002 15:09                2 325 liste.CS
07/05/2002 15:09                4 608 liste.exe
20/08/2001 11:54               86 016 ARTICLES.MDB

dos>liste articles.mdb

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                        2500 10 5
b300 pompe                       56 62 45
c300 arc                         3500 10 20
d300 flèches - lot de 6          780 12 20
e300 combinaison de plongée     2800 34 7
f300 bouteilles d'oxygène        800 10 5

```

## 6.3.4 Mise à jour d'une table

Les exemples précédents se contentaient de lister le contenu d'une table. Nous modifions notre programme de gestion de la base d'articles afin qu'il puisse modifier celle-ci. Le programme s'appelle *sql*. On lui passe en paramètre le nom DSN de la base d'articles à gérer. L'utilisateur tape directement des commandes SQL au clavier que le programme exécute comme le montrent les résultats qui suivent obtenus sur la base MySQL d'articles :

```
dos>vbc /r:microsoft.data.odbc.dll sql.vb

dos>sql mysql-articles

Requête SQL (fin pour arrêter) : select * from articles

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                                2500 10 5
b300 pompe                                56 62 45
c300 arc                                  3500 10 20
d300 flèches - lot de 6                   780 12 20
e300 combinaison de plongée              2800 34 7
f300 bouteilles d'oxygène                 800 10 5

Requête SQL (fin pour arrêter) : select * from articles where stock_actuel<stock_minimum

-----
code,nom,prix,stock_actuel,stock_minimum
-----

c300 arc                                  3500 10 20
d300 flèches - lot de 6                   780 12 20

Requête SQL (fin pour arrêter) : insert into articles values ("1","1",1,1,1)
Il y a eu 1 ligne(s) modifiée(s)

Requête SQL (fin pour arrêter) : select * from articles

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                                2500 10 5
b300 pompe                                56 62 45
c300 arc                                  3500 10 20
d300 flèches - lot de 6                   780 12 20
e300 combinaison de plongée              2800 34 7
f300 bouteilles d'oxygène                 800 10 5
1 1 1 1 1

Requête SQL (fin pour arrêter) : update articles set nom="2" where nom="1"
Il y a eu 1 ligne(s) modifiée(s)

Requête SQL (fin pour arrêter) : select * from articles

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                                2500 10 5
b300 pompe                                56 62 45
c300 arc                                  3500 10 20
d300 flèches - lot de 6                   780 12 20
e300 combinaison de plongée              2800 34 7
f300 bouteilles d'oxygène                 800 10 5
1 2 1 1 1

Requête SQL (fin pour arrêter) : delete from articles where code="1"
Il y a eu 1 ligne(s) modifiée(s)

Requête SQL (fin pour arrêter) : select * from articles

-----
code,nom,prix,stock_actuel,stock_minimum
-----

a300 vélo                                2500 10 5
b300 pompe                                56 62 45
c300 arc                                  3500 10 20
d300 flèches - lot de 6                   780 12 20
```

```
e300 combinaison de plongée      2800 34 7
f300 bouteilles d'oxygène        800 10 5
```

Requête SQL (fin pour arrêter) : select \* from articles order by nom asc

```
-----
code,nom,prix,stock_actuel,stock_minimum
-----
```

```
c300 arc                          3500 10 20
f300 bouteilles d'oxygène        800 10 5
e300 combinaison de plongée    2800 34 7
d300 flèches - lot de 6        780 12 20
b300 pompe                       56 62 45
a300 vélo                        2500 10 5
```

Requête SQL (fin pour arrêter) : fin

Le programme est le suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.Data.Odbc
Imports System.Data.OleDb
Imports System.Text.RegularExpressions
Imports System.Collections
Imports Microsoft.VisualBasic

Module db3
Public Sub Main(ByVal args() As String)

    ' application console
    ' exécute des requêtes SQL tapées au clavier sur une
    ' table ARTICLES d'une base DSN dont le nom est passé en paramètre
    Const syntaxe As String = "syntaxe : pg dsnArticles"

    ' vérification des paramètres
    ' a-t-on 2 paramètres
    If args.Length <> 1 Then
        ' msg d'erreur
        Console.Error.WriteLine(syntaxe)
        ' fin
        Environment.Exit(1)
    End If

    ' on récupère le paramètre
    Dim dsnArticles As String = args(0)
    ' chaîne de connexion à la base
    Dim connectString As String = "DSN=" + dsnArticles + ";"

    ' préparation de la connexion à la bd
    Dim articlesConn As OdbcConnection = Nothing
    Dim sqlCommand As OdbcCommand = Nothing
    Try
        ' on tente d'accéder à la base de données
        articlesConn = New OdbcConnection(connectString)
        articlesConn.Open()
        ' on crée un objet command
        sqlCommand = New OdbcCommand("", articlesConn)
    'try
    Catch ex As Exception
        ' msg d'erreur
        Console.Error.WriteLine(("Erreur d'exploitation de la base de données (" + ex.Message + ")"))
        ' libération des ressources
        Try
            articlesConn.Close()
        Catch
        End Try
        Environment.Exit(2)
    End Try

    ' on construit un dictionnaire des commandes sql acceptées
    Dim commandesSQL() As String = {"select", "insert", "update", "delete"}
    Dim dicoCommandes As New Hashtable
    Dim i As Integer
    For i = 0 To commandesSQL.Length - 1
        dicoCommandes.Add(commandesSQL(i), True)
    Next i
```

```

' lecture-exécution des commandes SQL tapées au clavier
Dim requête As String = Nothing ' texte de la requête SQL
Dim champs() As String ' les champs de la requête
Dim modèle As New Regex("\s+")
' boucle de saisie-exécution des commandes SQL tapées au clavier
While True
    ' pas d'erreur au départ
    Dim erreur As Boolean = False
    ' demande de la requête
    Console.Out.Write(ControlChars.Lf + "Requête SQL (fin pour arrêter) : ")
    requête = Console.In.ReadLine().Trim().ToLower()
    ' fini ?
    If requête = "fin" Then
        Exit While
    End If
    ' on décompose la requête en champs
    champs = modèle.Split(requête)
    ' requête valide ?
    If champs.Length = 0 Or Not dicoCommandes.ContainsKey(champs(0)) Then
        ' msg d'erreur
        Console.Error.WriteLine("Requête invalide. Utilisez select, insert, update, delete")
        ' requête suivante
        erreur = True
    End If
    If Not erreur Then
        ' préparation de l'objet Command pour exécuter la requête
        sqlCommand.CommandText = requête
        ' exécution de la requête
        Try
            If champs(0) = "select" Then
                executeSelect(sqlCommand)
            Else
                executeUpdate(sqlCommand)
            End If
        Catch ex As Exception
            ' msg d'erreur
            Console.Error.WriteLine(("Erreur d'exploitation de la base de données (" + ex.Message + ")"))
        End Try
    End If
End While
' libération des ressources
Try
    articlesConn.Close()
Catch
End Try
Environment.Exit(0)
End Sub

' exécution d'une requête de mise à jour
Sub executeUpdate(ByVal sqlCommand As OdbcCommand)
    ' exécute sqlCommand, requête de mise à jour
    Dim nbLignes As Integer = sqlCommand.ExecuteNonQuery()
    ' affichage
    Console.Out.WriteLine("Il y a eu " & nbLignes & " ligne(s) modifiée(s)")
End Sub

' exécution d'une requête Select
Sub executeSelect(ByVal sqlCommand As OdbcCommand)
    ' exécute sqlCommand, requête select
    Dim myReader As OdbcDataReader = sqlCommand.ExecuteReader()
    ' Exploitation de la table récupérée
    ' affichage des colonnes
    Dim ligne As String = ""
    Dim i As Integer
    For i = 0 To (myReader.FieldCount - 1) - 1
        ligne += myReader.GetName(i) + ", "
    Next i
    ligne += myReader.GetName(i)
    Console.Out.WriteLine((ControlChars.Lf + "".PadLeft(ligne.Length, "-"c) + ControlChars.Lf + ligne +
    ControlChars.Lf + "".PadLeft(ligne.Length, "-"c) + ControlChars.Lf))
    ' affichage des données
    While myReader.Read()
        ' exploitation ligne courante
        ligne = ""
        For i = 0 To myReader.FieldCount - 1
            ligne += myReader(i).ToString + " "
        Next i
        ' affichage
        Console.WriteLine(ligne)
    End While
End Sub

```

```

' libération des ressources
myReader.Close()
End Sub
End Module

```

Nous ne commentons ici que ce qui est nouveau par rapport au programme précédent :

- Nous construisons un dictionnaire des commandes sql acceptées :

```

' on construit un dictionnaire des commandes sql acceptées
Dim commandesSQL() As String = {"select", "insert", "update", "delete"}
Dim dicoCommandes As New Hashtable
Dim i As Integer
For i = 0 To commandesSQL.Length - 1
    dicoCommandes.Add(commandesSQL(i), True)
Next i

```

ce qui nous permet ensuite de vérifier simplement si le 1er mot (*champs[0]*) de la requête tapée est l'une des quatre commandes acceptées :

```

' requête valide ?
If champs.Length = 0 Or Not dicoCommandes.ContainsKey(champs(0)) Then
    ' msg d'erreur
    Console.Error.WriteLine("Requête invalide. Utilisez select, insert, update, delete")
    ' requête suivante
    erreur = True
End If 'if

```

- Auparavant la requête avait été décomposée en champs à l'aide de la méthode *Split* de la classe *Regex* :

```

Dim modèle As New Regex("\s+")
....
' on décompose la requête en champs
champs = modèle.Split(requête)

```

Les mots composant la requête peuvent être séparés d'un nombre quelconque d'espaces.

- L'exécution d'une requête *select* n'utilise pas la même méthode que celle d'une requête d'une mise à jour (*insert, update, delete*). Aussi doit-on faire un test et exécuter une fonction différente pour chacun de ces deux cas :

```

' préparation de l'objet Command pour exécuter la requête
sqlCommand.CommandText = requête
' exécution de la requête
Try
    If champs(0) = "select" Then
        executeSelect(sqlCommand)
    Else
        executeUpdate(sqlCommand)
    End If 'try
Catch ex As Exception
    ' msg d'erreur
    Console.Error.WriteLine(("Erreur d'exploitation de la base de données (" + ex.Message + ")"))
End Try

```

L'exécution d'une requête SQL peut générer une exception qui est ici gérée.

- La fonction *executeSelect* reprend tout ce qui a été vu dans les exemples précédents.
- La fonction *executeUpdate* utilise la méthode *ExecuteNonQuery* de la class *OdbcCommand* qui rend le nombre de lignes affectées par la commande.

## 6.3.5 IMPÔTS

Nous reprenons l'objet *impôt* construit dans un chapitre précédent :

```

' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System

Public Class impôt
    ' les données nécessaires au calcul de l'impôt
    ' proviennent d'une source extérieure

```

Accès aux bases de données

```

Private limites(), coeffR(), coeffN() As Decimal

' constructeur
Public Sub New(ByVal LIMITES() As Decimal, ByVal COEFFR() As Decimal, ByVal COEFFN() As Decimal)
' on vérifie que les 3 tableaux ont la même taille
Dim OK As Boolean = LIMITES.Length = COEFFR.Length And LIMITES.Length = COEFFN.Length
If Not OK Then
Throw New Exception("Les 3 tableaux fournis n'ont pas la même taille(" & LIMITES.Length & "," &
COEFFR.Length & "," & COEFFN.Length & ")")
End If
' c'est bon
Me.limites = LIMITES
Me.coeffR = COEFFR
Me.coeffN = COEFFN
End Sub

' calcul de l'impôt
Public Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Integer)
As Long
' calcul du nombre de parts
Dim nbParts As Decimal
If marié Then
nbParts = CDec(nbEnfants) / 2 + 2
Else
nbParts = CDec(nbEnfants) / 2 + 1
End If
If nbEnfants >= 3 Then
nbParts += 0.5D
End If
' calcul revenu imposable & Quotient familial
Dim revenu As Decimal = 0.72D * salaire
Dim QF As Decimal = revenu / nbParts
' calcul de l'impôt
limites((limites.Length - 1)) = QF + 1
Dim i As Integer = 0
While QF > limites(i)
i += 1
End While
' retour résultat
Return CLng(revenu * coeffR(i) - nbParts * coeffN(i))
End Function
End Class

```

Nous lui ajoutons un nouveau constructeur permettant d'initialiser les tableaux *limites*, *coeffR*, *coeffN* à partir d'une base de données ODBC :

```

Imports System.Data
Imports Microsoft.Data.Odbc
Imports System.Collections
...

```

```

' constructeur 2
Public Sub New(ByVal DSNimpots As String, ByVal Timpots As String, ByVal collimites As String, ByVal
colCoeffR As String, ByVal colCoeffN As String)
' initialise les trois tableaux limites, coeffR, coeffN à partir
' du contenu de la table Timpots de la base ODBC DSNimpots
' collimites, colCoeffR, colCoeffN sont les trois colonnes de cette table
' peut lancer une exception
Dim connectString As String = "DSN=" + DSNimpots + ";" ' chaîne de connexion à la base
Dim impotsConn As OdbcConnection = Nothing ' la connexion
Dim sqlCommand As OdbcCommand = Nothing ' la commande SQL
' la requête SELECT
Dim selectCommand As String = "select " + collimites + "," + colCoeffR + "," + colCoeffN + " from " +
Timpots
' tableaux pour récupérer les données
Dim tLimites As New ArrayList
Dim tCoeffR As New ArrayList
Dim tCoeffN As New ArrayList

' on tente d'accéder à la base de données
impotsConn = New OdbcConnection(connectString)
impotsConn.Open()
' on crée un objet command
sqlCommand = New OdbcCommand(selectCommand, impotsConn)
' on exécute la requête
Dim myReader As OdbcDataReader = sqlCommand.ExecuteReader()
' Exploitation de la table récupérée
While myReader.Read()
' les données de la ligne courante sont mis dans les tableaux

```

```

tLimites.Add(myReader(colLimites))
tCoeffR.Add(myReader(colCoeffR))
tCoeffN.Add(myReader(colCoeffN))
End While
' libération des ressources
myReader.Close()
impotsConn.Close()

' les tableaux dynamiques sont mis dans des tableaux statiques
Me.limites = New Decimal(tLimites.Count) {}
Me.coeffR = New Decimal(tCoeffR.Count) {}
Me.coeffN = New Decimal(tCoeffN.Count) {}
Dim i As Integer
For i = 0 To tLimites.Count - 1
    limites(i) = Decimal.Parse(tLimites(i).ToString())
    coeffR(i) = Decimal.Parse(tCoeffR(i).ToString())
    coeffN(i) = Decimal.Parse(tCoeffN(i).ToString())
Next i
End Sub

```

Le programme de test est le suivant : il reçoit en arguments les paramètres à transmettre au constructeur de la classe *impôt*. Après avoir construit un objet *impôt*, il fait des calculs d'impôt à payer :

```

Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

' pg de test
Module testimpots
Sub Main(ByVal arguments() As String)
    ' programme interactif de calcul d'impôt
    ' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
    ' le programme affiche alors l'impôt à payer
    Const syntaxe1 As String = "pg DSNimpots tabImpots collimites colCoeffR colCoeffN"
    Const syntaxe2 As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire : salaire annuel en F"

    ' vérification des paramètres du programme
    If arguments.Length <> 5 Then
        ' msg d'erreur
        Console.Error.WriteLine(syntaxe1)
        ' fin
        Environment.Exit(1)
    End If 'if
    ' on récupère les arguments
    Dim DSNimpots As String = arguments(0)
    Dim tabImpots As String = arguments(1)
    Dim collimites As String = arguments(2)
    Dim colCoeffR As String = arguments(3)
    Dim colCoeffN As String = arguments(4)

    ' création d'un objet impôt
    Dim objImpôt As impôt = Nothing
    Try
        objImpôt = New impôt(DSNimpots, tabImpots, collimites, colCoeffR, colCoeffN)
    Catch ex As Exception
        Console.Error.WriteLine("L'erreur suivante s'est produite : " + ex.Message)
        Environment.Exit(2)
    End Try

    ' boucle infinie
    While True
        ' au départ pas d'erreurs
        Dim erreur As Boolean = False

        ' on demande les paramètres du calcul de l'impôt
        Console.Out.Write("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :")
        Dim paramètres As String = Console.In.ReadLine().Trim()

        ' qq chose à faire ?
        If paramètres Is Nothing Or paramètres = "" Then
            Exit While
        End If

        ' vérification du nombre d'arguments dans la ligne saisie

```

```

Dim args As String() = paramètres.Split(Nothing)
Dim nbParamètres As Integer = args.Length
If nbParamètres <> 3 Then
    Console.Error.WriteLine(syntaxe2)
    erreur = True
End If
Dim marié As String
Dim nbEnfants As Integer
Dim salaire As Integer
If Not erreur Then
    ' vérification de la validité des paramètres
    ' marié
    marié = args(0).ToLower()
    If marié <> "o" And marié <> "n" Then
        Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument marié incorrect : tapez o ou
n"))
        erreur = True
    End If
    ' nbEnfants
    nbEnfants = 0
    Try
        nbEnfants = Integer.Parse(args(1))
        If nbEnfants < 0 Then
            Throw New Exception
        End If
    Catch
        Console.Error.WriteLine(syntaxe2 + "\nArgument nbEnfants incorrect : tapez un entier positif ou
nul")
        erreur = True
    End Try
    ' salaire
    salaire = 0
    Try
        salaire = Integer.Parse(args(2))
        If salaire < 0 Then
            Throw New Exception
        End If
    Catch
        Console.Error.WriteLine(syntaxe2 + "\nArgument salaire incorrect : tapez un entier positif ou
nul")
        erreur = True
    End Try
End If
If Not erreur Then
    ' les paramètres sont corrects - on calcule l'impôt
    Console.Out.WriteLine(("impôt=" & objImpôt.calculer(marié = "o", nbEnfants, salaire).ToString + "
F"))
End If
End While
End Sub
End Module

```

La base utilisée est une base MySQL de nom DSN *mysql-impots* :

```

C:\mysql\bin>mysql --database=impots --user=admimpots --password=mdpimpots
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.49-max-debug

Type 'help' for help.

mysql> show tables;
+-----+
| Tables_in_impots |
+-----+
| timpots          |
+-----+

mysql> select * from timpots;
+-----+-----+-----+
| limites | coeffR | coeffN |
+-----+-----+-----+
| 12620   | 0       | 0       |
| 13190   | 0.05    | 631     |
| 15640   | 0.1     | 1290.5  |
| 24740   | 0.15    | 2072.5  |
| 31810   | 0.2     | 3309.5  |
| 39970   | 0.25    | 4900    |
| 48360   | 0.3     | 6898    |
| 55790   | 0.35    | 9316.5  |
| 92970   | 0.4     | 12106   |
| 127860  | 0.45    | 16754   |
+-----+-----+-----+

```

	151250		0.5		23147.5	
	172040		0.55		30710	
	195000		0.6		39312	
	0		0.65		49062	
+-----+-----+-----+						

L'exécution du programme de test donne les résultats suivants :

```

dos>D:\data\devel\vbnet\poly\chap6\impots>vbc /r:system.data.dll /r:microsoft.data.odbc.dll /r:system.dll
/t:library impots.vb

dos>vbc /r:impots.dll testimpots.vb

dos>test mysql-impots timpots limites coeffr coeffn
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 200000
impôt=22506 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 2 200000
impôt=33388 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 3 200000
impôt=16400 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 3 300000
impôt=50082 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 3 200000
impôt=22506 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :

```

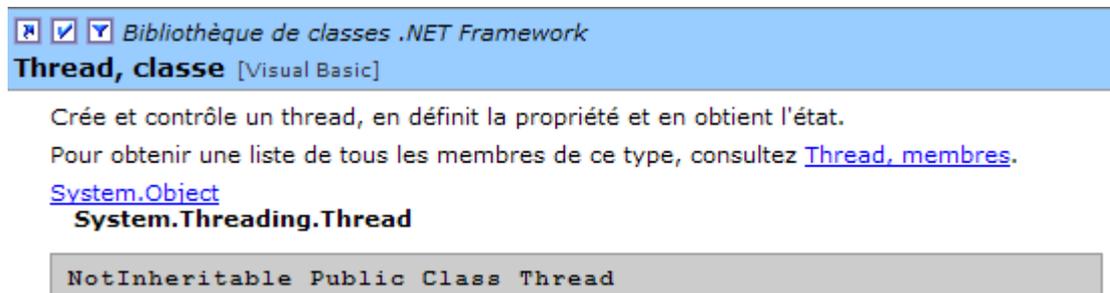
## 6.4 Accès aux données en mode déconnecté

Ce sujet sera traité ultérieurement.

# 7. Les threads d'exécution

## 7.1 Introduction

Lorsqu'on lance une application, elle s'exécute dans un flux d'exécution appelé un **thread**. La classe .NET modélisant un *thread* est la classe *System.Threading.Thread* et a la définition suivante :



Bibliothèque de classes .NET Framework  
**Thread, classe** [Visual Basic]  
Crée et contrôle un thread, en définit la propriété et en obtient l'état.  
Pour obtenir une liste de tous les membres de ce type, consultez [Thread, membres](#).  
[System.Object](#)  
**System.Threading.Thread**  
NotInheritable Public Class Thread

Nous n'utiliserons que certaines des propriétés et méthodes de cette classe :

CurrentThread - propriété statique	donne le thread actuellement en cours d'exécution
Name - propriété d'objet	nom du thread
isAlive - propriété d'objet	indique si le thread est actif(true) ou non (false)
Start - méthode d'objet	lance l'exécution d'un thread
Abort - méthode d'objet	arrête définitivement l'exécution d'un thread
Sleep(n) - méthode statique	arrête l'exécution d'un thread pendant n millisecondes
Suspend() - méthode d'objet	suspend temporairement l'exécution d'un thread
Resume() - méthode d'objet	repréend l'exécution d'un thread suspendu
Join() - méthode d'objet	opération bloquante - attend la fin du thread pour passer à l'instruction suivante

Regardons une première application mettant en évidence l'existence d'un thread principal d'exécution, celui dans lequel s'exécute la fonction *Main* d'une classe :

```
' utilisation de threads
Imports System
Imports System.Threading

Public Module thread1
    Public Sub Main()
        ' init thread courant
        Dim main As Thread = Thread.CurrentThread
        ' affichage
        Console.Out.WriteLine("Thread courant : " + main.Name)
        ' on change le nom
        main.Name = "main"
        ' vérification
        Console.Out.WriteLine("Thread courant : " + main.Name)
        ' boucle infinie
        While True
            ' affichage
            Console.Out.WriteLine((main.Name + " : " + DateTime.Now.ToString("hh:mm:ss")))
            ' arrêt temporaire
            Thread.Sleep(1000)
        End While
    End Sub
End Module
```

Les résultats écran :

```
dos>thread1
Thread courant :
Les threads d'exécution
```

```
Thread courant : main
main : 06:13:55
main : 06:13:56
main : 06:13:57
main : 06:13:58
main : 06:13:59
^C
```

L'exemple précédent illustre les points suivants :

- la fonction *Main* s'exécute bien dans un thread
- on a accès aux caractéristiques de ce thread par *Thread.CurrentThread*
- le rôle de la méthode *Sleep*. Ici le thread exécutant *Main* se met en sommeil régulièrement pendant 1 seconde entre deux affichages.

## 7.2 Création de threads d'exécution

Il est possible d'avoir des applications où des morceaux de code s'exécutent de façon "simultanée" dans différents threads d'exécution. Lorsqu'on dit que des *threads* s'exécutent de façon simultanée, on commet souvent un abus de langage. Si la machine n'a qu'un processeur comme c'est encore souvent le cas, les *threads* se partagent ce processeur : ils en disposent, chacun leur tour, pendant un court instant (quelques millisecondes). C'est ce qui donne l'illusion du parallélisme d'exécution. La portion de temps accordée à un *thread* dépend de divers facteurs dont sa priorité qui a une valeur par défaut mais qui peut être fixée également par programmation. Lorsqu'un *thread* dispose du processeur, il l'utilise normalement pendant tout le temps qui lui a été accordé. Cependant, il peut le libérer avant terme :

- en se mettant en attente d'un événement (*wait, join, suspend*)
- en se mettant en sommeil pendant un temps déterminé (*sleep*)

1. Un **thread T** est d'abord créé par son constructeur

```
Public Sub New(ByVal start As ThreadStart)
```

*ThreadStart* est de type delegate et définit le prototype d'une fonction sans paramètres :

```
Public Delegate Sub ThreadStart()
```

Une construction classique est la suivante :

```
dim T as Thread=new Thread(new ThreadStart(run));
```

La fonction *run* passée en paramètres sera exécutée au lancement du Thread.

2. L'exécution du thread T est lancé par **T.Start()** : la fonction [run] passée au constructeur de T va alors être exécutée par le thread T. Le programme qui exécute l'instruction *T.start()* n'attend pas la fin de la tâche T : il passe aussitôt à l'instruction qui suit. On a alors deux tâches qui s'exécutent en parallèle. Elles doivent souvent pouvoir communiquer entre elles pour savoir où en est le travail commun à réaliser. C'est le problème de synchronisation des threads.
3. Une fois lancé, le thread s'exécute de façon autonome. Il s'arrêtera lorsque la fonction *start* qu'il exécute aura fini son travail.
4. On peut envoyer certains signaux à la tâche T :
  - a. **T.Suspend()** lui dit de s'arrêter momentanément
  - b. **T.Resume()** lui dit de reprendre son travail
  - c. **T.Abort()** lui dit de s'arrêter définitivement
5. On peut aussi attendre la fin de son exécution par **T.join()**. On a là une instruction bloquante : le programme qui l'exécute est bloqué jusqu'à ce que la tâche T ait terminé son travail. C'est un moyen de synchronisation.

Examinons le programme suivant :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Threading

Module thread2
  Public Sub Main()
    ' init Thread courant
```

```

Dim main As Thread = Thread.CurrentThread
' on fixe un nom au Thread
main.Name = "main"

' création de threads d'exécution
Dim tâches(4) As Thread
Dim i As Integer
For i = 0 To tâches.Length - 1
    ' on crée le thread i
    tâches(i) = New Thread(New ThreadStart(AddressOf affiche))
    ' on fixe le nom du thread
    tâches(i).Name = "tache_" & i
    ' on lance l'exécution du thread i
    tâches(i).Start()
Next i
' fin de main
Console.Out.WriteLine("fin du thread " + main.Name)
End Sub

Public Sub affiche()
    ' affichage début d'exécution
    Console.Out.WriteLine("Début d'exécution de la méthode affiche dans le Thread " +
Thread.CurrentThread.Name + " : " + DateTime.Now.ToString("hh:mm:ss"))
    ' mise en sommeil pendant 1 s
    Thread.Sleep(1000)
    ' affichage fin d'exécution
    Console.Out.WriteLine("Fin d'exécution de la méthode affiche dans le Thread " +
Thread.CurrentThread.Name + " : " + DateTime.Now.ToString("hh:mm:ss"))
End Sub
End Module

```

Le thread principal, celui qui exécute la fonction *Main*, crée 5 autres threads chargés d'exécuter la méthode statique *affiche*. Les résultats sont les suivants :

```

dos>thread2
fin du thread main
Début d'exécution de la méthode affiche dans le Thread tache_0 : 05:27:53
Début d'exécution de la méthode affiche dans le Thread tache_1 : 05:27:53
Début d'exécution de la méthode affiche dans le Thread tache_2 : 05:27:53
Début d'exécution de la méthode affiche dans le Thread tache_3 : 05:27:53
Début d'exécution de la méthode affiche dans le Thread tache_4 : 05:27:53
Fin d'exécution de la méthode affiche dans le Thread tache_0 : 05:27:54
Fin d'exécution de la méthode affiche dans le Thread tache_1 : 05:27:54
Fin d'exécution de la méthode affiche dans le Thread tache_2 : 05:27:54
Fin d'exécution de la méthode affiche dans le Thread tache_3 : 05:27:54
Fin d'exécution de la méthode affiche dans le Thread tache_4 : 05:27:54

```

Ces résultats sont très instructifs :

- on voit tout d'abord que le lancement de l'exécution d'un thread n'est pas bloquante. La méthode *Main* a lancé l'exécution de 5 threads en parallèle et a terminé son exécution avant eux. L'opération

```

' on lance l'exécution du thread i
tâches(i).Start()

```

lance l'exécution du thread *tâches[i]* mais ceci fait, l'exécution se poursuit immédiatement avec l'instruction qui suit sans attendre la fin d'exécution du thread.

- tous les threads créés doivent exécuter la méthode *affiche*. L'ordre d'exécution est imprévisible. Même si dans l'exemple, l'ordre d'exécution semble suivre l'ordre des demandes d'exécution, on ne peut en conclure de généralités. Le système d'exploitation a ici 6 threads et un processeur. Il va distribuer le processeur à ces 6 threads selon des règles qui lui sont propres.
- on voit dans les résultats une conséquence de la méthode *Sleep*. dans l'exemple, c'est le thread 0 qui exécute le premier la méthode *affiche*. Le message de début d'exécution est affiché puis il exécute la méthode *Sleep* qui le suspend pendant 1 seconde. Il perd alors le processeur qui devient ainsi disponible pour un autre thread. L'exemple montre que c'est le thread 1 qui va l'obtenir. Le thread 1 va suivre le même parcours ainsi que les autres threads. Lorsque la seconde de sommeil du thread 0 va être terminée, son exécution peut reprendre. Le système lui donne le processeur et il peut terminer l'exécution de la méthode *affiche*.

Modifions notre programme pour le terminer la méthode *Main* par les instructions :

```

' fin de main
Console.Out.WriteLine("fin du thread " + main.Name)
Environment.Exit(0)

```

L'exécution du nouveau programme donne :

```
fin du thread main
```

Les threads créés par la fonction *Main* ne sont pas exécutés. C'est l'instruction

```
Environment.Exit(0)
```

qui fait cela : elle supprime tous les threads de l'application et non simplement le thread *Main*. La solution à ce problème est que la méthode *Main* attende la fin d'exécution des threads qu'elle a créés avant de se terminer elle-même. Cela peut se faire avec la méthode *Join* de la classe *Thread* :

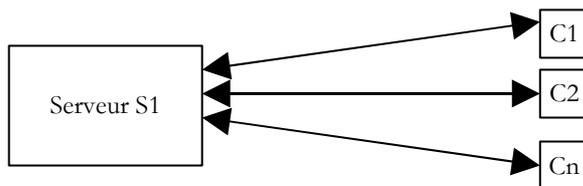
```
' on attend la fin d'exécution de tous les threads
For i = 0 To tâches.Length - 1
    ' attente de la fin d'exécution du thread i
    tâches(i).Join()
Next i 'for
' fin de main
Console.Out.WriteLine("fin du thread " + main.Name)
Environment.Exit(0)
```

On obtient alors les résultats suivants :

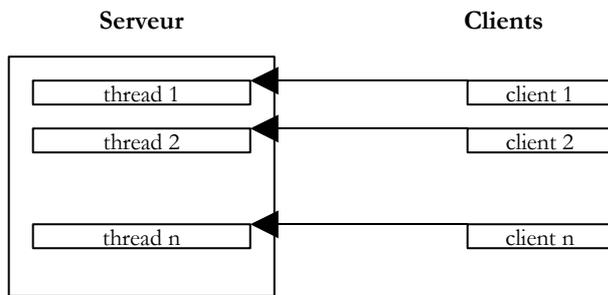
```
Début d'exécution de la méthode affiche dans le Thread tache_1 : 05:34:48
Début d'exécution de la méthode affiche dans le Thread tache_2 : 05:34:48
Début d'exécution de la méthode affiche dans le Thread tache_3 : 05:34:48
Début d'exécution de la méthode affiche dans le Thread tache_4 : 05:34:48
Début d'exécution de la méthode affiche dans le Thread tache_0 : 05:34:48
Fin d'exécution de la méthode affiche dans le Thread tache_2 : 05:34:50
Fin d'exécution de la méthode affiche dans le Thread tache_1 : 05:34:50
Fin d'exécution de la méthode affiche dans le Thread tache_3 : 05:34:50
Fin d'exécution de la méthode affiche dans le Thread tache_0 : 05:34:50
Fin d'exécution de la méthode affiche dans le Thread tache_4 : 05:34:50
fin du thread main
```

## 7.3 Intérêt des threads

Maintenant que nous avons mis en évidence l'existence d'un thread par défaut, celui qui exécute la méthode *Main*, et que nous savons comment en créer d'autres, arrêtons-nous sur l'intérêt pour nous des threads et sur la raison pour laquelle nous les présentons ici. Il y a un type d'applications qui se prêtent bien à l'utilisation des threads, ce sont les applications client-serveur de l'internet. Dans une telle application, un serveur situé sur une machine S1 répond aux demandes de clients situés sur des machines distantes C1, C2, ..., Cn.



Nous utilisons tous les jours des applications de l'internet correspondant à ce schéma : services Web, messagerie électronique, consultation de forums, transfert de fichiers... Dans le schéma ci-dessus, le serveur S1 doit servir les clients Ci de façon simultanée. Si nous prenons l'exemple d'un serveur FTP (File Transfer Protocol) qui délivre des fichiers à ses clients, nous savons qu'un transfert de fichier peut prendre parfois plusieurs heures. Il est bien sûr hors de question qu'un client monopolise tout seul le serveur une telle durée. Ce qui est fait habituellement, c'est que le serveur crée autant de threads d'exécution qu'il y a de clients. Chaque thread est alors chargé de s'occuper d'un client particulier. Le processeur étant partagé cycliquement entre tous les threads actifs de la machine, le serveur passe alors un peu de temps avec chaque client assurant ainsi la simultanéité du service.



## 7.4 Accès à des ressources partagées

Dans l'exemple client-serveur évoqué ci-dessus, chaque thread sert un client de façon largement indépendante. Néanmoins, les threads peuvent être amenés à coopérer pour rendre le service demandé à leur client notamment pour l'accès à des ressources partagées. Le schéma ci-dessus fait penser aux guichets d'une grande administration, une poste par exemple où à chaque guichet un agent sert un client. Supposons que de temps en temps ces agents soient amenés à faire des photocopies de documents amenés par leurs clients et qu'il n'y ait qu'une photocopieuse. Deux agents ne peuvent utiliser la photocopieuse en même temps. Si l'agent  $i$  trouve la photocopieuse utilisée par l'agent  $j$ , il devra attendre. On appelle cette situation, l'accès à une ressource partagée et en informatique elle est assez délicate à gérer. Prenons l'exemple suivant :

- une application va générer  $n$  threads,  $n$  étant passé en paramètre
- la ressource partagée est un compteur qui devra être incrémenté par chaque thread généré
- à la fin de l'application, la valeur du compteur est affiché. On devrait donc trouver  $n$ .

Le programme est le suivant :

```
' options
Option Explicit On
Option Strict On

' utilisation de threads
Imports System
Imports System.Threading

Public Class thread3
' variables de classe
Private Shared cptrThreads As Integer = 0

Public Overloads Shared Sub Main(ByVal args() As [String])
' mode d'emploi
Const syntaxe As String = "pg nbThreads"
Const nbMaxThreads As Integer = 100

' vérification nbre d'arguments
If args.Length <> 1 Then
' erreur
Console.Error.WriteLine(syntaxe)
' arrêt
Environment.Exit(1)
End If
' vérification qualité de l'argument
Dim nbThreads As Integer = 0
Try
nbThreads = Integer.Parse(args(0))
If nbThreads < 1 Or nbThreads > nbMaxThreads Then
Throw New Exception
End If
Catch
' erreur
Console.Error.WriteLine("Nombre de threads incorrect (entre 1 et " & nbMaxThreads & ")")
' fin
Environment.Exit(2)
End Try
' création et génération des threads
Dim threads(nbThreads - 1) As Thread
Dim i As Integer
For i = 0 To nbThreads - 1
' création
threads(i) = New Thread(New ThreadStart(AddressOf incremente))
' nommage
threads(i).Name = "tache_" & i
```

Les threads d'exécution

```

    ' lancement
    threads(i).Start()
Next i
' attente de la fin des threads
For i = 0 To nbThreads - 1
    threads(i).Join()
Next i ' affichage compteur
Console.Out.WriteLine("Nombre de threads générés : " & cptrThreads)
End Sub

Public Shared Sub incrémente()
    ' augmente le compteur de threads
    ' lecture compteur
    Dim valeur As Integer = cptrThreads
    ' suivi
    Console.Out.WriteLine("A " + DateTime.Now.ToString("hh:mm:ss") & ", le thread " &
Thread.CurrentThread.Name & " a lu la valeur du compteur : " & cptrThreads)
    ' attente
    Thread.Sleep(1000)
    ' incrémentation compteur
    cptrThreads = valeur + 1
    ' suivi
    Console.Out.WriteLine("A " & DateTime.Now.ToString("hh:mm:ss") & ", le thread " &
Thread.CurrentThread.Name & " a écrit la valeur du compteur : " & cptrThreads)
End Sub
End Class

```

Nous ne nous attarderons pas sur la partie génération de threads déjà étudiée. Intéressons-nous plutôt à la méthode *incrémente*, utilisée par chaque thread pour incrémenter le compteur statique *cptrThreads*.

1. le compteur est lu
2. le thread s'arrête 1 s. Il perd donc le processeur
3. le compteur est incrémenté

L'étape 2 n'est là que pour forcer le thread à perdre le processeur. Celui-ci va être donné à un autre thread. Dans la pratique, rien n'assure qu'un thread ne sera pas interrompu entre le moment où il va lire le compteur et le moment où il va l'incrémenter. Le risque existe de perdre le processeur entre le moment où on lit la valeur du compteur et celui où on écrit sa valeur incrémentée de 1. En effet, l'opération d'incrémenter va faire l'objet de plusieurs instructions élémentaires au niveau du processeur qui peuvent être interrompues. L'étape 2 de sommeil d'une seconde n'est donc là que pour systématiser ce risque. Les résultats obtenus sont les suivants :

```

dos>thread3 5
A 05:44:34, le thread tache_0 a lu la valeur du compteur : 0
A 05:44:34, le thread tache_1 a lu la valeur du compteur : 0
A 05:44:34, le thread tache_2 a lu la valeur du compteur : 0
A 05:44:34, le thread tache_3 a lu la valeur du compteur : 0
A 05:44:34, le thread tache_4 a lu la valeur du compteur : 0
A 05:44:35, le thread tache_0 a écrit la valeur du compteur : 1
A 05:44:35, le thread tache_1 a écrit la valeur du compteur : 1
A 05:44:35, le thread tache_2 a écrit la valeur du compteur : 1
A 05:44:35, le thread tache_3 a écrit la valeur du compteur : 1
A 05:44:35, le thread tache_4 a écrit la valeur du compteur : 1
Nombre de threads générés : 1

```

A la lecture de ces résultats, on voit bien ce qui se passe :

- un premier thread lit le compteur. Il trouve 0.
- il s'arrête 1 s donc perd le processeur
- un second thread prend alors le processeur et lit lui aussi la valeur du compteur. Elle est toujours à 0 puisque le thread précédent ne l'a pas encore incrémenté. Il s'arrête lui aussi 1 s.
- en 1 s, les 5 threads ont le temps de passer tous et de lire tous la valeur 0.
- lorsqu'ils vont se réveiller les uns après les autres, ils vont incrémenter la valeur 0 qu'ils ont lue et écrire la valeur 1 dans le compteur, ce que confirme le programme principal (Main).

D'où vient le problème ? Le second thread a lu une mauvaise valeur du fait que le premier avait été interrompu avant d'avoir terminé son travail qui était de mettre à jour le compteur dans la fenêtre. Cela nous amène à la notion de ressource critique et de section critique d'un programme:

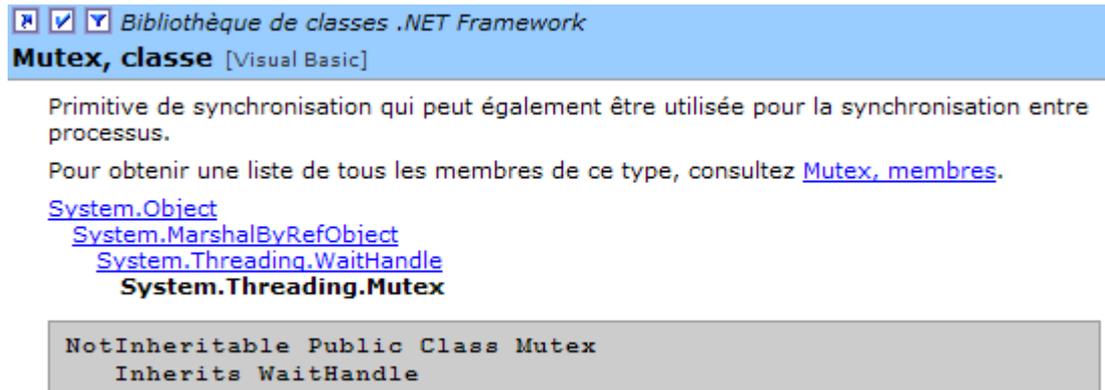
- une ressource critique est une ressource qui ne peut être détenue que par un thread à la fois. Ici la ressource critique est le compteur.
- une section critique d'un programme est une séquence d'instructions dans le flux d'exécution d'un thread au cours de laquelle il accède à une ressource critique. On doit assurer qu'au cours de cette section critique, il est le seul à avoir accès à la ressource.

## 7.5 Accès exclusif à une ressource partagée

Dans notre exemple, la section critique est le code situé entre la lecture du compteur et l'écriture de sa nouvelle valeur :

```
' lecture compteur
Dim valeur As Integer = cptrThreads
' attente
Thread.Sleep(1000)
' incrémentation compteur
cptrThreads = valeur + 1
```

Pour exécuter ce code, un thread doit être assuré d'être tout seul. Il peut être interrompu mais pendant cette interruption, un autre thread ne doit pas pouvoir exécuter ce même code. La plate-forme .NET offre plusieurs outils pour assurer l'entrée unitaire dans les sections critiques de code. Nous utiliserons la classe **Mutex** :



**Bibliothèque de classes .NET Framework**  
**Mutex, classe** [Visual Basic]

Primitive de synchronisation qui peut également être utilisée pour la synchronisation entre processus.

Pour obtenir une liste de tous les membres de ce type, consultez [Mutex, membres](#).

[System.Object](#)  
[System.MarshalByRefObject](#)  
[System.Threading.WaitHandle](#)  
**System.Threading.Mutex**

```
NotInheritable Public Class Mutex
    Inherits WaitHandle
```

Nous n'utiliserons ici que les constructeurs et méthodes suivants :

<code>public Mutex()</code>	crée un objet de synchronisation M
<code>public bool WaitOne()</code>	Le thread T1 qui exécute l'opération <code>M.WaitOne()</code> demande la propriété de l'objet de synchronisation M. Si le Mutex M n'est détenu par aucun thread (le cas au départ), il est "donné" au thread T1 qui l'a demandé. Si un peu plus tard, un thread T2 fait la même opération, il sera bloqué. En effet, un Mutex ne peut appartenir qu'à un thread. Il sera débloqué lorsque le thread T1 libèrera le mutex M qu'il détient. Plusieurs threads peuvent ainsi être bloqués en attente du Mutex M.
<code>public void ReleaseMutex()</code>	Le thread T1 qui effectue l'opération <code>M.ReleaseMutex()</code> abandonne la propriété du Mutex M. Lorsque le thread T1 perdra le processeur, le système pourra le donner à l'un des threads en attente du Mutex M. Un seul l'obtiendra à son tour, les autres en attente de M restant bloqués

Un Mutex M gère l'accès à une ressource partagée R. Un thread demande la ressource R par `M.WaitOne()` et la rend par `M.ReleaseMutex()`. Une section critique de code qui ne doit être exécutée que par un seul thread à la fois est une ressource partagée. La synchronisation d'exécution de la section critique peut se faire ainsi :

```
M.WaitOne()
' le thread est seul à entrer ici
' section critique
....
M.ReleaseMutex()
```

où M est un objet *Mutex*. Il faut bien sûr ne jamais oublier de libérer un *Mutex* devenu inutile pour qu'un autre thread puisse entrer dans la section critique, sinon les threads en attente d'un Mutex jamais libéré n'auront jamais accès au processeur. Par ailleurs, il faut éviter la situation d'interblocage (*deadlock*) dans laquelle deux threads s'attendent mutuellement. Considérons les actions suivantes qui se suivent dans le temps :

- un thread T1 obtient la propriété d'un Mutex M1 pour avoir accès à une ressource partagée R1
- un thread T2 obtient la propriété d'un Mutex M2 pour avoir accès à une ressource partagée R2
- le thread T1 demande le Mutex M2. Il est bloqué.
- le thread T2 demande le Mutex M1. Il est bloqué.

Ici, les threads T1 et T2 s'attendent mutuellement. Ce cas apparaît lorsque des threads ont besoin de deux ressources partagées, la ressource R1 contrôlée par le Mutex M1 et la ressource R2 contrôlée par le Mutex M2. Une solution possible est de demander les deux ressources en même temps à l'aide d'un Mutex unique M. Mais ce n'est pas toujours possible si par exemple cela entraîne une

Les threads d'exécution

mobilisation longue d'une ressource coûteuse. Une autre solution est qu'un thread ayant M1 et ne pouvant obtenir M2, relâche alors M1 pour éviter l'interblocage. Si nous mettons en pratique ce que nous venons de voir sur l'exemple précédent, notre application devient la suivante :

```
' options
Option Explicit On
Option Strict On

' utilisation de threads
Imports System
Imports System.Threading

Public Class thread4
' variables de classe
Private Shared cptrThreads As Integer = 0 ' compteur de threads
Private Shared autorisation As Mutex

Public Overloads Shared Sub Main(ByVal args() As [String])
' mode d'emploi
Const syntaxe As String = "pg nbThreads"
Const nbMaxThreads As Integer = 100

' vérification nbre d'arguments
If args.Length <> 1 Then
' erreur
Console.Error.WriteLine(syntaxe)
' arrêt
Environment.Exit(1)
End If
' vérification qualité de l'argument
Dim nbThreads As Integer = 0
Try
nbThreads = Integer.Parse(args(0))
If nbThreads < 1 Or nbThreads > nbMaxThreads Then
Throw New Exception
End If
Catch
End Try

' initialisation de l'autorisation d'accès à une section critique
autorisation = New Mutex

' création et génération des threads
Dim threads(nbThreads) As Thread
Dim i As Integer
For i = 0 To nbThreads - 1
' création
threads(i) = New Thread(New ThreadStart(AddressOf incremente))
' nommage
threads(i).Name = "tache_" & i
' lancement
threads(i).Start()
Next i
' attente de la fin des threads
For i = 0 To nbThreads - 1
threads(i).Join()
Next i
' affichage compteur
Console.Out.WriteLine("Nombre de threads générés : " & cptrThreads)
End Sub

Public Shared Sub incremente()
' augmente le compteur de threads
' on demande l'autorisation d'entrer dans la section critique
autorisation.WaitOne()
' lecture compteur
Dim valeur As Integer = cptrThreads
' suivi
Console.Out.WriteLine("A " & DateTime.Now.ToString("hh:mm:ss") & ", le thread " &
Thread.CurrentThread.Name & " a lu la valeur du compteur : " & cptrThreads)
' attente
Thread.Sleep(1000)
' incrémentation compteur
cptrThreads = valeur + 1
' suivi
Console.Out.WriteLine("A " & DateTime.Now.ToString("hh:mm:ss") & ", le thread " &
Thread.CurrentThread.Name & " a écrit la valeur du compteur : " & cptrThreads)
' on rend l'autorisation d'accès
autorisation.ReleaseMutex()
End Sub
```

Les résultats obtenus sont conformes à ce qui était attendu :

```
dos>thread4 5
A 05:51:10, le thread tache_0 a lu la valeur du compteur : 0
A 05:51:11, le thread tache_0 a écrit la valeur du compteur : 1
A 05:51:11, le thread tache_1 a lu la valeur du compteur : 1
A 05:51:12, le thread tache_1 a écrit la valeur du compteur : 2
A 05:51:12, le thread tache_2 a lu la valeur du compteur : 2
A 05:51:13, le thread tache_2 a écrit la valeur du compteur : 3
A 05:51:13, le thread tache_3 a lu la valeur du compteur : 3
A 05:51:14, le thread tache_3 a écrit la valeur du compteur : 4
A 05:51:14, le thread tache_4 a lu la valeur du compteur : 4
A 05:51:15, le thread tache_4 a écrit la valeur du compteur : 5
Nombre de threads générés : 5
```

## 7.6 Synchronisation par événements

Considérons la situation suivante, appelée parfois situation de **producteurs-consommateurs**.

1. On a un tableau dans lequel des processus viennent déposer des données (les producteurs) et d'autres viennent les lire (les consommateurs).
2. Les producteurs sont égaux entre-eux mais exclusifs : un seul producteur à la fois peut déposer ses données dans le tableau.
3. Les consommateurs sont égaux entre-eux mais exclusifs : un seul lecteur à la fois peut lire les données déposées dans le tableau.
4. Un consommateur ne peut lire les données du tableau que lorsqu'un producteur en a déposé dedans et un producteur ne peut déposer de nouvelles données dans le tableau que lorsque celles qui y sont ont été consommées.

On peut dans cet exposé distinguer deux ressources partagées :

1. le tableau en écriture
2. le tableau en lecture

L'accès à ces deux ressources partagées peut être contrôlée par des Mutex comme vu précédemment, un pour chaque ressource. Une fois qu'un consommateur a obtenu le tableau en lecture, il doit vérifier qu'il y a bien des données dedans. On utilisera un événement pour l'en avertir. De même un producteur ayant obtenu le tableau en écriture devra attendre qu'un consommateur l'ait vidé. On utilisera là encore un événement.

Les événements utilisés feront partie de la classe *AutoResetEvent* :

**Bibliothèque de classes .NET Framework**

**AutoResetEvent, classe** [Visual Basic]

Avertit un ou thread en attente qu'un événement s'est produit. Cette classe ne peut pas être héritée.

Pour obtenir une liste de tous les membres de ce type, consultez [AutoResetEvent, membres](#).

[System.Object](#)  
[System.MarshalByRefObject](#)  
[System.Threading.WaitHandle](#)  
**System.Threading.AutoResetEvent**

```
NotInheritable Public Class AutoResetEvent
    Inherits WaitHandle
```

Ce type d'événement est analogue à un booléen mais évite des attentes actives ou semi-actives. Ainsi si le droit d'écriture est contrôlé par un booléen *peutEcrire*, un producteur avant d'écrire exécutera un code du genre :

```
while (peutEcrire==false) ' attente active
```

ou

```
while (peutEcrire==false) ' attente semi-active
    Thread.Sleep(100) ' attente de 100ms
end while
```

Dans la première méthode, le thread mobilise inutilement le processeur. Dans la seconde, il vérifie l'état du booléen *peutEcrire* toutes les 100 ms. La classe *AutoResetEvent* permet encore d'améliorer les choses : le thread va demander à être réveillé lorsque l'événement qu'il attend se sera produit :

```
AutoEvent peutEcrire=new AutoResetEvent(false) ' peutEcrire=false;
....
peutEcrire.WaitOne() ' le thread attend que l'évt peutEcrire passe à vrai
```

L'opération

```
AutoEvent peutEcrire=new AutoResetEvent(false) ' peutEcrire=false;
```

initialise le booléen *peutEcrire* à *false*. L'opération

```
peutEcrire.WaitOne() ' le thread attend que l'évt peutEcrire passe à vrai
```

exécutée par un thread fait que celui-ci passe si le booléen *peutEcrire* est vrai ou sinon est bloqué jusqu'à ce qu'il devienne vrai. Un autre thread le passera à vrai par l'opération *peutEcrire.Set()* ou à faux par l'opération *peutEcrire.Reset()*.

Le programme de producteurs-consommateurs est le suivant :

```
' utilisation de threads lecteurs et écrivains
' illustre l'utilisation simultanée de ressources partagées et de synchronisation

' options
Option Explicit On
Option Strict On

' utilisation de threads
Imports System
Imports System.Threading

Public Class lececr

    ' variables de classe
    Private Shared data(5) As Integer ' ressource partagée entre threads lecteur et threads écrivain
    Private Shared lecteur As Mutex ' variable de synchronisation pour lire le tableau
    Private Shared écrivain As Mutex ' variable de synchronisation pour écrire dans le tableau
    Private Shared objRandom As New Random(DateTime.Now.Second) ' un générateur de nombres aléatoires
    Private Shared peutLire As AutoResetEvent ' signale qu'on peut lire le contenu de data
    Private Shared peutEcrire As AutoResetEvent

    Public Shared Sub Main(ByVal args() As [String])

        ' le nbre de threads à générer
        Const nbThreads As Integer = 3

        ' initialisation des drapeaux
        peutLire = New AutoResetEvent(False) ' on ne peut pas encore lire
        peutEcrire = New AutoResetEvent(True) ' on peut déjà écrire

        ' initialisation des variables de synchronisation
        lecteur = New Mutex ' synchronise les lecteurs
        écrivain = New Mutex ' synchronise les écrivains

        ' création des threads lecteurs
        Dim lecteurs(nbThreads) As Thread
        Dim i As Integer
        For i = 0 To nbThreads - 1
            ' création
            lecteurs(i) = New Thread(New ThreadStart(AddressOf lire))
            lecteurs(i).Name = "lecteur_" & i
            ' lancement
            lecteurs(i).Start()
        Next i

        ' création des threads écrivains
        Dim écrivains(nbThreads) As Thread
        For i = 0 To nbThreads - 1
            ' création
            écrivains(i) = New Thread(New ThreadStart(AddressOf écrire))
            écrivains(i).Name = "écrivain_" & i
            ' lancement
            écrivains(i).Start()
        Next i

        ' fin de main
        Console.Out.WriteLine("fin de Main...")
    End Sub
End Class
```

Les threads d'exécution

```

End Sub

' lire le contenu du tableau
Public Shared Sub lire()
    ' section critique
    lecteur.WaitOne() ' un seul lecteur peut passer
    peutLire.WaitOne() ' on doit pouvoir lire

    ' lecture tableau
    Dim i As Integer
    For i = 0 To data.Length - 1
        'attente 1 s
        Thread.Sleep(1000)
        ' affichage
        Console.Out.WriteLine((DateTime.Now.ToString("hh:mm:ss") & " : Le lecteur " &
Thread.CurrentThread.Name & " a lu le nombre " & data(i)))
    Next i

    ' on ne peut plus lire
    peutLire.Reset()
    ' on peut écrire
    peutEcrire.Set()
    ' fin de section critique
    lecteur.ReleaseMutex()
End Sub

' écrire dans le tableau
Public Shared Sub écrire()
    ' section critique
    ' un seul écrivain peut passer
    écrivain.WaitOne()
    ' on doit attendre l'autorisation d'écriture
    peutEcrire.WaitOne()

    ' écriture tableau
    Dim i As Integer
    For i = 0 To data.Length - 1
        'attente 1 s
        Thread.Sleep(1000)
        ' affichage
        data(i) = objRandom.Next(0, 1000)
        Console.Out.WriteLine((DateTime.Now.ToString("hh:mm:ss") & " : L'écrivain " &
Thread.CurrentThread.Name & " a écrit le nombre " & data(i)))
    Next i

    ' on ne peut plus écrire
    peutEcrire.Reset()
    ' on peut lire
    peutLire.Set()
    'fin de section critique
    écrivain.ReleaseMutex()
End Sub
End Class

```

L'exécution donne les résultats suivants :

```

dos>lececr
fin de Main...
05:56:56 : L'écrivain écrivain_0 a écrit le nombre 459
05:56:57 : L'écrivain écrivain_0 a écrit le nombre 955
05:56:58 : L'écrivain écrivain_0 a écrit le nombre 212
05:56:59 : L'écrivain écrivain_0 a écrit le nombre 297
05:57:00 : L'écrivain écrivain_0 a écrit le nombre 37
05:57:01 : L'écrivain écrivain_0 a écrit le nombre 623
05:57:02 : Le lecteur lecteur_0 a lu le nombre 459
05:57:03 : Le lecteur lecteur_0 a lu le nombre 955
05:57:04 : Le lecteur lecteur_0 a lu le nombre 212
05:57:05 : Le lecteur lecteur_0 a lu le nombre 297
05:57:06 : Le lecteur lecteur_0 a lu le nombre 37
05:57:07 : Le lecteur lecteur_0 a lu le nombre 623
05:57:08 : L'écrivain écrivain_1 a écrit le nombre 549
05:57:09 : L'écrivain écrivain_1 a écrit le nombre 34
05:57:10 : L'écrivain écrivain_1 a écrit le nombre 781
05:57:11 : L'écrivain écrivain_1 a écrit le nombre 555
05:57:12 : L'écrivain écrivain_1 a écrit le nombre 812
05:57:13 : L'écrivain écrivain_1 a écrit le nombre 406
05:57:14 : Le lecteur lecteur_1 a lu le nombre 549
05:57:15 : Le lecteur lecteur_1 a lu le nombre 34
05:57:16 : Le lecteur lecteur_1 a lu le nombre 781
05:57:17 : Le lecteur lecteur_1 a lu le nombre 555
05:57:18 : Le lecteur lecteur_1 a lu le nombre 812

```

Les threads d'exécution

```
05:57:19 : Le lecteur lecteur_1 a lu le nombre 406
05:57:20 : L'écrivain écrivain_2 a écrit le nombre 442
05:57:21 : L'écrivain écrivain_2 a écrit le nombre 83
^C
```

On peut remarquer les points suivants :

- on a bien 1 seul lecteur à la fois bien que celui-ci perde le processeur dans la section critique *lire*
- on a bien 1 seul écrivain à la fois bien que celui-ci perde le processeur dans la section critique *écrire*
- un lecteur ne lit que lorsqu'il y a quelque chose à lire dans le tableau
- un écrivain n'écrit que lorsque le tableau a été entièrement lu

# 8. Programmation TCP-IP

## 8.1 Généralités

### 8.1.1 Les protocoles de l'Internet

Nous donnons ici une introduction aux protocoles de communication de l'Internet, appelés aussi suite de protocoles **TCP/IP** (*Transfer Control Protocol / Internet Protocol*), du nom des deux principaux protocoles. Il est bon que le lecteur ait une compréhension globale du fonctionnement des réseaux et notamment des protocoles TCP/IP avant d'aborder la construction d'applications distribuées.

Le texte qui suit est une traduction partielle d'un texte que l'on trouve dans le document "*Lan Workplace for Dos - Administrator's Guide*" de NOVELL, document du début des années 90.

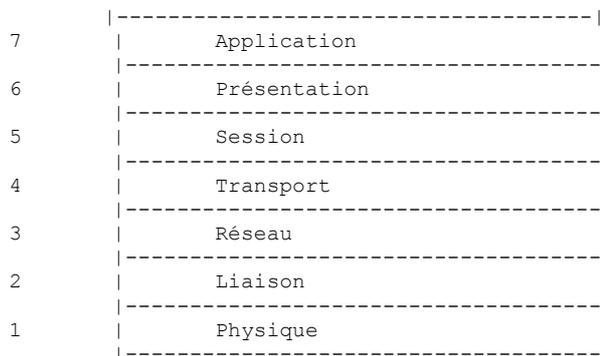
Le concept général de créer un réseau d'ordinateurs hétérogènes vient de recherches effectuées par le **DARPA** (**D**efense **A**dvanced **R**esearch **P**rojects **A**gency) aux Etats-Unis. Le DARPA a développé la suite de protocoles connue sous le nom de TCP/IP qui permet à des machines hétérogènes de communiquer entre elles. Ces protocoles ont été testés sur un réseau appelé **ARPAnet**, réseau qui devint ultérieurement le réseau **INTERNET**. Les protocoles TCP/IP définissent des formats et des règles de transmission et de réception indépendants de l'organisation des réseaux et des matériels utilisés.

Le réseau conçu par le DARPA et géré par les protocoles TCP/IP est un réseau à **commutation de paquets**. Un tel réseau transmet l'information sur le réseau, en petits morceaux appelés **paquets**. Ainsi, si un ordinateur transmet un gros fichier, ce dernier sera découpé en petits morceaux qui seront envoyés sur le réseau pour être recomposés à destination. TCP/IP définit le format de ces paquets, à savoir :

- . origine du paquet
- . destination
- . longueur
- . type

### 8.1.2 Le modèle OSI

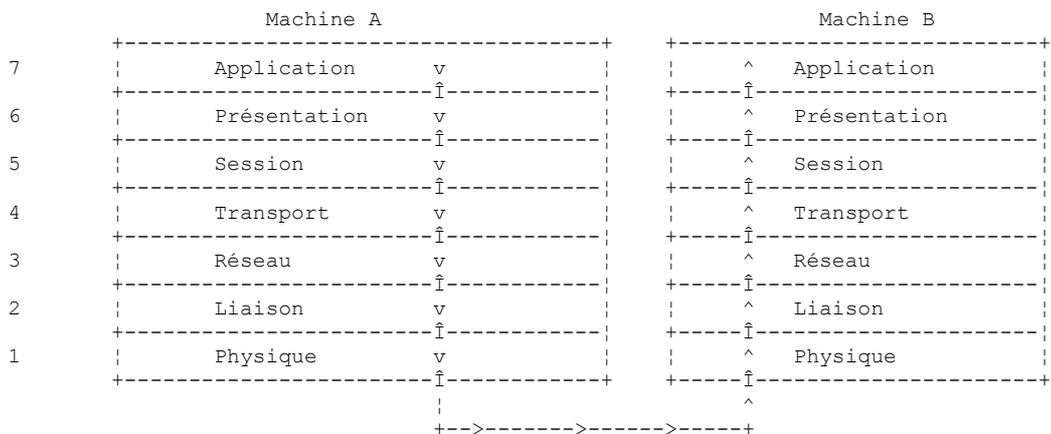
Les protocoles TCP/IP suivent à peu près le modèle de réseau ouvert appelé **OSI** (**O**pen **S**ystems **I**nterconnection **R**eference **M**odel) défini par l'**ISO** (**I**nternational **S**tandards **O**rganisation). Ce modèle décrit un réseau idéal où la communication entre machines peut être représentée par un modèle à sept couches :



Chaque couche reçoit des services de la couche inférieure et offre les siens à la couche supérieure. Supposons que deux applications situées sur des machines A et B différentes veulent communiquer : elles le font au niveau de la couche *Application*. Elles n'ont pas besoin de connaître tous les détails du fonctionnement du réseau : chaque application remet l'information qu'elle souhaite transmettre à la couche du dessous : la couche *Présentation*. L'application n'a donc à connaître que les règles d'interfaçage avec la couche *Présentation*.

Une fois l'information dans la couche *Présentation*, elle est passée selon d'autres règles à la couche *Session* et ainsi de suite, jusqu'à ce que l'information arrive sur le support physique et soit transmise physiquement à la machine destination. Là, elle subira le traitement inverse de celui qu'elle a subi sur la machine expéditeur.

A chaque couche, le processus expéditeur chargé d'envoyer l'information, l'envoie à un processus récepteur sur l'autre machine appartenant à la même couche que lui. Il le fait selon certaines règles que l'on appelle le **protocole** de la couche. On a donc le schéma de communication final suivant :

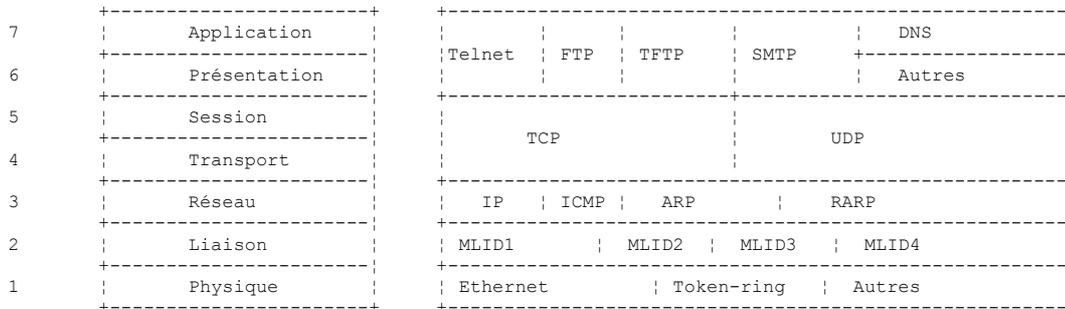


Le rôle des différentes couches est le suivant :

Physique	Assure la transmission de bits sur un support physique. On trouve dans cette couche des équipements terminaux de traitement des données (E.T.T.D.) tels que terminal ou ordinateur, ainsi que des équipements de terminaison de circuits de données (E.T.C.D.) tels que modulateur/démodulateur, multiplexeur, concentrateur. Les points d'intérêt à ce niveau sont : <ul style="list-style-type: none"> <li>. le choix du codage de l'information (analogique ou numérique)</li> <li>. le choix du mode de transmission (synchrone ou asynchrone).</li> </ul>
Liaison de données	Masque les particularités physiques de la couche Physique. Détecte et corrige les erreurs de transmission.
Réseau	Gère le chemin que doivent suivre les informations envoyées sur le réseau. On appelle cela le <i>routing</i> : déterminer la route à suivre par une information pour qu'elle arrive à son destinataire.
Transport	Permet la communication entre deux applications alors que les couches précédentes ne permettraient que la communication entre machines. Un service fourni par cette couche peut être le multiplexage : la couche transport pourra utiliser une même connexion réseau (de machine à machine) pour transmettre des informations appartenant à plusieurs applications.
Session	On va trouver dans cette couche des services permettant à une application d'ouvrir et de maintenir une session de travail sur une machine distante.
Présentation	Elle vise à uniformiser la représentation des données sur les différentes machines. Ainsi des données provenant d'une machine A, vont être "habillées" par la couche <i>Présentation</i> de la machine A, selon un format standard avant d'être envoyées sur le réseau. Parvenues à la couche <i>Présentation</i> de la machine destinatrice B qui les reconnaîtra grâce à leur format standard, elles seront habillées d'une autre façon afin que l'application de la machine B les reconnaisse.
Application	A ce niveau, on trouve les applications généralement proches de l'utilisateur telles que la messagerie électronique ou le transfert de fichiers.

### 8.1.3 Le modèle TCP/IP

Le modèle OSI est un modèle idéal encore jamais réalisé. La suite de protocoles TCP/IP s'en approche sous la forme suivante :



## Couche Physique

En réseau local, on trouve généralement une technologie **Ethernet** ou **Token-Ring**. Nous ne présentons ici que la technologie Ethernet.

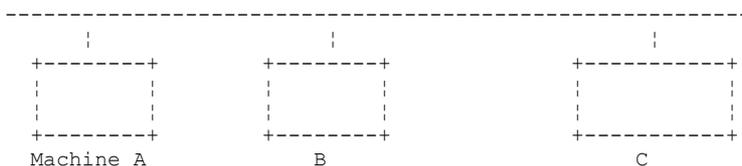
## Ethernet

C'est le nom donné à une technologie de réseaux locaux à commutation de paquets inventée à PARC Xerox au début des années 1970 et normalisée par Xerox, Intel et Digital Equipment en 1978. Le réseau est physiquement constitué d'un câble coaxial d'environ 1,27 cm de diamètre et d'une longueur de 500 m au plus. Il peut être étendu au moyen de *répéteurs*, deux machines ne pouvant être séparées par plus de deux répéteurs. Le câble est passif : tous les éléments actifs sont sur les machines raccordées au câble. Chaque machine est reliée au câble par une carte d'accès au réseau comprenant :

- un transmetteur (*transceiver*) qui détecte la présence de signaux sur le câble et convertit les signaux analogiques en signaux numérique et inversement.
- un coupleur qui reçoit les signaux numériques du transmetteur et les transmet à l'ordinateur pour traitement ou inversement.

Les caractéristiques principales de la technologie Ethernet sont les suivantes :

- Capacité de 10 Mégabits/seconde.
- Topologie en bus : toutes les machines sont raccordées au même câble



- Réseau diffusant - Une machine qui émet transfère des informations sur le câble avec l'adresse de la machine destinatrice. Toutes les machines raccordées reçoivent alors ces informations et seule celle à qui elles sont destinées les conserve.
- La méthode d'accès est la suivante : le transmetteur désirant émettre écoute le câble - il détecte alors la présence ou non d'une onde porteuse, présence qui signifierait qu'une transmission est en cours. C'est la technique **CSMA** (*Carrier Sense Multiple Access*). En l'absence de porteuse, un transmetteur peut décider de transmettre à son tour. Ils peuvent être plusieurs à prendre cette décision. Les signaux émis se mélangent : on dit qu'il y a **collision**. Le transmetteur détecte cette situation : en même temps qu'il émet sur le câble, il écoute ce qui passe réellement sur celui-ci. S'il détecte que l'information transitant sur le câble n'est pas celle qu'il a émise, il en déduit qu'il y a collision et il s'arrêtera d'émettre. Les autres transmetteurs qui émettaient feront de même. Chacun reprendra son émission après un temps aléatoire dépendant de chaque transmetteur. Cette technique est appelée **CD** (*Collision Detect*). La méthode d'accès est ainsi appelée **CSMA/CD**.
- un adressage sur 48 bits. Chaque machine a une adresse, appelée ici adresse physique, qui est inscrite sur la carte qui la relie au câble. On appelle cet adresse, l'adresse *Ethernet* de la machine.

## Couche Réseau

Nous trouvons au niveau de cette couche, les protocoles IP, ICMP, ARP et RARP.



ICMP (Internet Control Message Protocol)	ICMP réalise la communication entre le programme du protocole IP d'une machine et celui d'une autre machine. C'est donc un protocole d'échange de messages à l'intérieur même du protocole IP.
ARP (Address Resolution Protocol)	fait la correspondance adresse Internet machine--> adresse physique machine
RARP (Reverse Address Resolution Protocol)	fait la correspondance adresse physique machine--> adresse Internet machine

### Couches Transport/Session

Dans cette couche, on trouve les protocoles suivants :

TCP (Transmission Control Protocol)	Assure une remise fiable d'informations entre deux clients
UDP (User Datagram Protocol)	Assure une remise non fiable d'informations entre deux clients

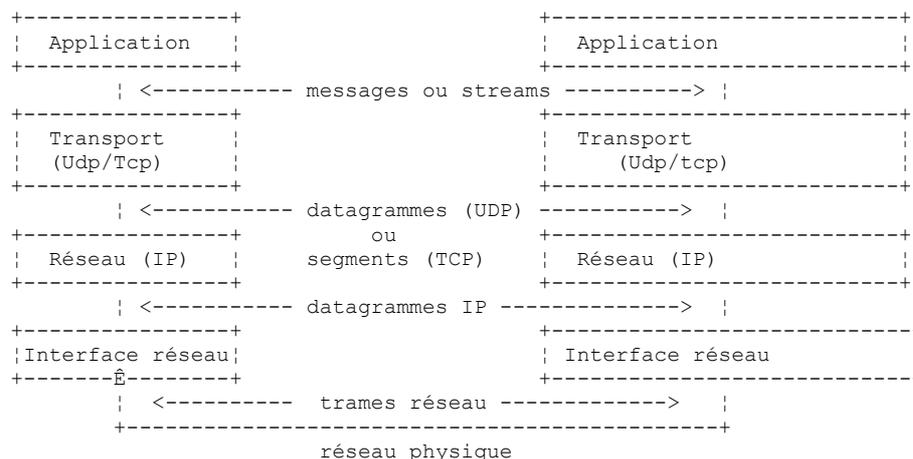
### Couches Application/Présentation/Session

On trouve ici divers protocoles :

TELNET	Emulateur de terminal permettant à une machine A de se connecter à une machine B en tant que terminal
FTP (File Transfer Protocol)	permet des transferts de fichiers
TFTP (Trivial File Transfer Protocol)	permet des transferts de fichiers
SMTP (Simple Mail Transfer protocol)	permet l'échange de messages entre utilisateurs du réseau
DNS (Domain Name System)	transforme un nom de machine en adresse Internet de la machine
XDR (eXternal Data Representation)	créé par sun Microsystems, il spécifie une représentation standard des données, indépendante des machines
RPC (Remote Procedures Call)	défini également par Sun, c'est un protocole de communication entre applications distantes, indépendant de la couche transport. Ce protocole est important : il décharge le programmeur de la connaissance des détails de la couche transport et rend les applications portables. Ce protocole s'appuie sur le protocole XDR
NFS (Network File System)	toujours défini par Sun, ce protocole permet à une machine, de "voir" le système de fichiers d'une autre machine. Il s'appuie sur le protocole RPC précédent

## 8.1.4 Fonctionnement des protocoles de l'Internet

Les applications développées dans l'environnement TCP/IP utilisent généralement plusieurs des protocoles de cet environnement. Un programme d'application communique avec la couche la plus élevée des protocoles. Celle-ci passe l'information à la couche du dessous et ainsi de suite jusqu'à arriver sur le support physique. Là, l'information est physiquement transférée à la machine destinataire où elle retraversera les mêmes couches, en sens inverse cette fois-ci, jusqu'à arriver à l'application destinataire des informations envoyées. Le schéma suivant montre le parcours de l'information :



Prenons un exemple : l'application FTP, définie au niveau de la couche *Application* et qui permet des transferts de fichiers entre machines.

- L'application délivre une suite d'octets à transmettre à la couche *transport*.
- La couche *transport* découpe cette suite d'octets en *segments* TCP, et ajoute au début de chaque segment, le numéro de celui-ci. Les segments sont passés à la couche Réseau gouvernée par le protocole **IP**.
- La couche IP crée un paquet encapsulant le segment TCP reçu. En tête de ce paquet, elle place les adresses Internet des machines source et destination. Elle détermine également l'adresse physique de la machine destinatrice. Le tout est passé à la couche *Liaison de données & Liaison physique*, c'est à dire à la carte réseau qui couple la machine au réseau physique.
- Là, le paquet IP est encapsulé à son tour dans une **trame** physique et envoyé à son destinataire sur le câble.
- Sur la machine destinatrice, la couche *Liaison de données & Liaison physique* fait l'inverse : elle désencapsule le paquet IP de la trame physique et le passe à la couche IP.
- La couche IP vérifie que le paquet est correct : elle calcule une somme, fonction des bits reçus (*checksum*), somme qu'elle doit retrouver dans l'en-tête du paquet. Si ce n'est pas le cas, celui-ci est rejeté.
- Si le paquet est déclaré correct, la couche IP désencapsule le segment TCP qui s'y trouve et le passe au-dessus à la couche *transport*.
- La couche *transport*, couche TCP dans notre exemple, examine le numéro du segment afin de restituer le bon ordre des segments.
- Elle calcule également une somme de vérification pour le segment TCP. S'il est trouvé correct, la couche TCP envoie un accusé de réception à la machine source, sinon le segment TCP est refusé.
- Il ne reste plus à la couche TCP qu'à transmettre la partie données du segment à l'application destinatrice de celles-ci dans la couche du dessus.

## 8.1.5 L'adressage dans l'Internet

Un *noeud* d'un réseau peut être un ordinateur, une imprimante intelligente, un serveur de fichiers, n'importe quoi en fait pouvant communiquer à l'aide des protocoles TCP/IP. Chaque noeud a une **adresse physique** ayant un format dépendant du type du réseau. Sur un réseau Ethernet, l'adresse physique est codée sur 6 octets. Une adresse d'un réseau X25 est un nombre à 14 chiffres.

L'**adresse Internet** d'un noeud est une adresse **logique** : elle est indépendante du matériel et du réseau utilisé. C'est une adresse sur 4 octets identifiant à la fois un réseau local et un noeud de ce réseau. L'adresse Internet est habituellement représentée sous la forme de 4 nombres, valeurs des 4 octets, séparés par un point. Ainsi l'adresse de la machine Lagaffe de la faculté des Sciences d'Angers est notée 193.49.144.1 et celle de la machine Liny 193.49.144.9. On en déduira que l'adresse Internet du réseau local est 193.49.144.0. On pourra avoir jusqu'à 254 noeuds sur ce réseau.

Parce que les adresses Internet ou adresses IP sont indépendantes du réseau, une machine d'un réseau A peut communiquer avec une machine d'un réseau B sans se préoccuper du type de réseau sur lequel elle se trouve : il suffit qu'elle connaisse son adresse IP. Le protocole IP de chaque réseau se charge de faire la conversion adresse IP <--> adresse physique, dans les deux sens.

Les adresses **IP** doivent être toutes différentes. En France, c'est l'INRIA qui s'occupe d'affecter les adresses IP. En fait, cet organisme délivre une adresse pour votre réseau local, par exemple 193.49.144.0 pour le réseau de la faculté des sciences d'Angers. L'administrateur de ce réseau peut ensuite affecter les adresses IP 193.49.144.1 à 193.49.144.254 comme il l'entend. Cette adresse est généralement inscrite dans un fichier particulier de chaque machine reliée au réseau.

### 8.1.5.1 Les classes d'adresses IP

Une adresse IP est une suite de 4 octets notée souvent I1.I2.I3.I4, qui contient en fait deux adresses :

- . l'adresse du réseau
- . l'adresse d'un noeud de ce réseau

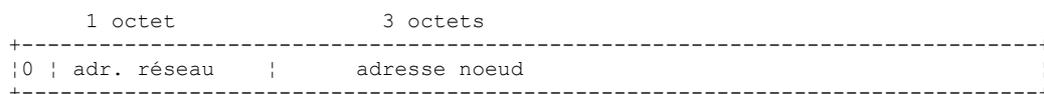
Selon la taille de ces deux champs, les adresses IP sont divisées en 3 classes : classes A, B et C.

#### Classe A

L'adresse IP : I1.I2.I3.I4 a la forme R1.N1.N2.N3 où

R1 est l'adresse du réseau  
N1.N2.N3 est l'adresse d'une machine dans ce réseau

Plus exactement, la forme d'une adresse IP de classe A est la suivante :



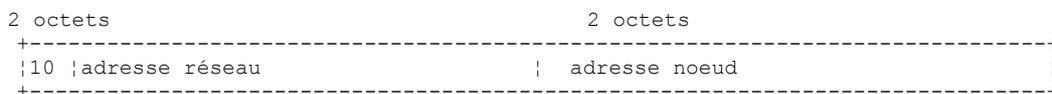
L'adresse réseau est sur 7 bits et l'adresse du noeud sur 24 bits. On peut donc avoir 127 réseaux de classe A, chacun comportant jusqu'à  $2^{24}$  noeuds.

#### Classe B

Ici, l'adresse IP : I1.I2.I3.I4 a la forme R1.R2.N1.N2 où

R1.R2 est l'adresse du réseau  
N1.N2 est l'adresse d'une machine dans ce réseau

Plus exactement, la forme d'une adresse IP de classe B est la suivante :



L'adresse du réseau est sur 2 octets (14 bits exactement) ainsi que celle du noeud. On peut donc avoir  $2^{14}$  réseaux de classe B chacun comportant jusqu'à  $2^{16}$  noeuds.

#### Classe C

Dans cette classe, l'adresse IP : I1.I2.I3.I4 a la forme R1.R2.R3.N1 où

R1.R2.R3 est l'adresse du réseau  
N1 est l'adresse d'une machine dans ce réseau

Plus exactement, la forme d'une adresse IP de classe C est la suivante :



L'adresse réseau est sur 3 octets (moins 3 bits) et l'adresse du noeud sur 1 octet. On peut donc avoir  $2^{21}$  réseaux de classe C comportant jusqu'à 256 noeuds.

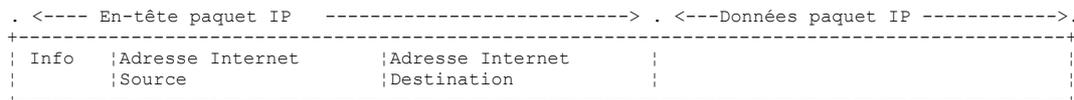
L'adresse de la machine *Lagaffe* de la faculté des sciences d'Angers étant 193.49.144.1, on voit que l'octet de poids fort vaut 193, c'est à dire en binaire 11000001. On en déduit que le réseau est de classe C.

#### Adresses réservées

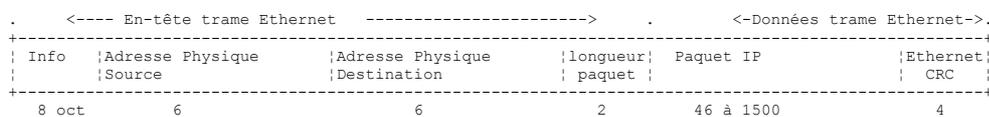
- Certaines adresses IP sont des adresses de réseaux plutôt que des adresses de noeuds dans le réseau. Ce sont celles, où l'adresse du noeud est mise à 0. Ainsi, l'adresse 193.49.144.0 est l'adresse IP du réseau de la Faculté des Sciences d'Angers. En conséquence, aucun noeud d'un réseau ne peut avoir l'adresse zéro.
- Lorsque dans une adresse IP, l'adresse du noeud ne comporte que des 1, on a alors une adresse de diffusion : cette adresse désigne **tous les noeuds du réseau**.
- Dans un réseau de classe C, permettant théoriquement  $2^8=256$  noeuds, si on enlève les deux adresses interdites, on n'a plus que 254 adresses autorisées.

### 8.1.5.2 Les protocoles de conversion Adresse Internet <--> Adresse physique

Nous avons vu que lors d'une émission d'informations d'une machine vers une autre, celles-ci à la traversée de la couche IP étaient encapsulées dans des paquets. Ceux-ci ont la forme suivante :



Le paquet IP contient donc les adresses Internet des machines source et destination. Lorsque ce paquet va être transmis à la couche chargée de l'envoyer sur le réseau physique, d'autres informations lui sont ajoutées pour former la trame physique qui sera finalement envoyée sur le réseau. Par exemple, le format d'une trame sur un réseau Ethernet est le suivant :



Dans la trame finale, il y a l'adresse physique des machines source et destination. Comment sont-elles obtenues ?

La machine expéditrice connaissant l'adresse IP de la machine avec qui elle veut communiquer obtient l'adresse physique de celle-ci en utilisant un protocole particulier appelé **ARP** (*Address Resolution Protocol*).

- Elle envoie un paquet d'un type spécial appelé paquet ARP contenant l'adresse IP de la machine dont on cherche l'adresse physique. Elle a pris soin également d'y placer sa propre adresse IP ainsi que son adresse physique.
- Ce paquet est envoyé à tous les noeuds du réseau.
- Ceux-ci reconnaissent la nature spéciale du paquet. Le noeud qui reconnaît son adresse IP dans le paquet, répond en envoyant à l'expéditeur du paquet son adresse physique. Comment le peut-il ? Il a trouvé dans le paquet les adresses IP et physique de l'expéditeur.
- L'expéditeur reçoit donc l'adresse physique qu'il cherchait. Il la stocke en mémoire afin de pouvoir l'utiliser ultérieurement si d'autres paquets sont à envoyer au même destinataire.

L'adresse IP d'une machine est normalement inscrite dans l'un de ses fichiers qu'elle peut donc consulter pour la connaître. Cette adresse peut être changée : il suffit d'éditer le fichier. L'adresse physique elle, est inscrite dans une mémoire de la carte réseau et ne peut être changée.

Lorsqu'un administrateur désire d'organiser son réseau différemment, il peut être amené à changer les adresses IP de tous les noeuds et donc à éditer les différents fichiers de configuration des différents noeuds. Cela peut être fastidieux et une occasion d'erreurs s'il y a beaucoup de machines. Une méthode consiste à ne pas affecter d'adresse IP aux machines : on inscrit alors un code spécial dans le fichier dans lequel la machine devrait trouver son adresse IP. Découvrant qu'elle n'a pas d'adresse IP, la machine la demande selon un protocole appelé **RARP** (*Reverse Address Resolution Protocol*). Elle envoie alors sur un réseau un paquet spécial appelé paquet RARP, analogue au paquet ARP précédent, dans lequel elle met son adresse physique. Ce paquet est envoyé à tous les noeuds qui reconnaissent alors un paquet RARP. L'un d'entre-eux, appelé **serveur RARP**, possède un fichier donnant la correspondance adresse physique <--> adresse IP de tous les noeuds. Il répond alors à l'expéditeur du paquet RARP, en lui renvoyant son adresse IP. Un administrateur désirant reconfigurer son réseau, n'a donc qu'à éditer le fichier de correspondances du serveur RARP. Celui-ci doit normalement avoir une adresse IP fixe qu'il doit pouvoir connaître sans avoir à utiliser lui-même le protocole RARP.

## 8.1.6 La couche réseau dite couche IP de l'internet

Le protocole IP (*Internet Protocol*) définit la forme que les paquets doivent prendre et la façon dont ils doivent être gérés lors de leur émission ou de leur réception. Ce type de paquet particulier est appelé un **datagramme IP**. Nous l'avons déjà présenté :

```

. <---- En-tête paquet IP -----> . <----Données paquet IP ----->.
+-----+
| Info   |Adresse Internet   |Adresse Internet   |
| Source |Destination        |
+-----+

```

L'important est qu'outre les données à transmettre, le datagramme IP contient les adresses Internet des machines source et destination. Ainsi la machine destinatrice sait qui lui envoie un message.

A la différence d'une trame de réseau qui a une longueur déterminée par les caractéristiques physiques du réseau sur lequel elle transite, la longueur du datagramme IP est elle fixée par le logiciel et sera donc la même sur différents réseaux physiques. Nous avons vu qu'en descendant de la couche réseau dans la couche physique le datagramme IP était encapsulé dans une trame physique. Nous avons donné l'exemple de la trame physique d'un réseau Ethernet :

```

. <---- En-tête trame Ethernet -----> . <----Données trame Ethernet----->.
+-----+
| Info   |Adresse Physique   |Adresse Physique   | Type du| Paquet IP   |Ethernet|
| Source |Destination        |Type du paquet    |
+-----+

```

Les trames physiques circulent de noeud en noeud vers leur destination qui peut ne pas être sur le même réseau physique que la machine expéditrice. Le paquet IP peut donc être encapsulé successivement dans des trames physiques différentes au niveau des noeuds qui font la jonction entre deux réseaux de type différent. Il se peut aussi que le paquet IP soit trop grand pour être encapsulé dans une trame physique. Le logiciel IP du noeud où se pose ce problème, décompose alors le paquet IP en *fragments* selon des règles précises, chacun d'eux étant ensuite envoyé sur le réseau physique. Ils ne seront réassemblés qu'à leur ultime destination.

### 8.1.6.1 Le routage

Le routage est la méthode d'acheminement des paquets IP à leur destination. Il y a deux méthodes : le routage direct et le routage indirect.

#### Routage direct

Le routage direct désigne l'acheminement d'un paquet IP directement de l'expéditeur au destinataire à l'intérieur du même réseau :

- . La machine expéditrice d'un datagramme IP a l'adresse IP du destinataire.
- . Elle obtient l'adresse physique de ce dernier par le protocole ARP ou dans ses tables, si cette adresse a déjà été obtenue.
- . Elle envoie le paquet sur le réseau à cette adresse physique.

#### Routage indirect

Le routage indirect désigne l'acheminement d'un paquet IP à une destination se trouvant sur un autre réseau que celui auquel appartient l'expéditeur. Dans ce cas, les parties adresse réseau des adresses IP des machines source et destination sont différentes. La machine source reconnaît ce point. Elle envoie alors le paquet à un noeud spécial appelé **routeur** (*router*), noeud qui connecte un réseau local aux autres réseaux et dont elle trouve l'adresse IP dans ses tables, adresse obtenue initialement soit dans un fichier soit dans une mémoire permanente ou encore via des informations circulant sur le réseau.

Un **routeur** est attaché à deux réseaux et possède une adresse IP à l'intérieur de ces deux réseaux.

```

      +-----+
réseau 2 | routeur  | réseau 1
-----+-----+-----
193.49.145.0 |193.49.144.6| 193.49.144.0
      |193.49.145.3|
      +-----+

```

Dans notre exemple ci-dessus :

- . Le réseau n° 1 a l'adresse Internet 193.49.144.0 et le réseau n° 2 l'adresse 193.49.145.0.
- . A l'intérieur du réseau n° 1, le routeur a l'adresse 193.49.144.6 et l'adresse 193.49.145.3 à l'intérieur du réseau n° 2.

Le routeur a pour rôle de mettre le paquet IP qu'il reçoit et qui est contenu dans une trame physique typique du réseau n° 1, dans une trame physique pouvant circuler sur le réseau n° 2. Si l'adresse IP du destinataire du paquet est dans le réseau n° 2, le routeur lui enverra le paquet directement sinon il l'enverra à un autre routeur, connectant le réseau n° 2 à un réseau n° 3 et ainsi de suite.

### 8.1.6.2 Messages d'erreur et de contrôle

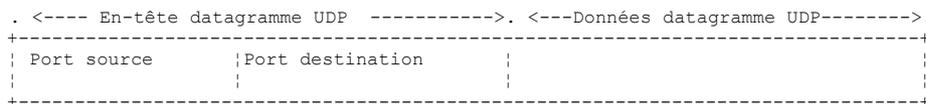
Toujours dans la couche réseau, au même niveau donc que le protocole IP, existe le protocole **ICMP** (*Internet Control Message Protocol*). Il sert à envoyer des messages sur le fonctionnement interne du réseau : noeuds en panne, embouteillage à un routeur, etc ... Les messages ICMP sont encapsulés dans des paquets IP et envoyés sur le réseau. Les couches IP des différents noeuds prennent les actions appropriées selon les messages ICMP qu'elles reçoivent. Ainsi, une application elle-même, ne voit jamais ces problèmes propres au réseau. Un noeud utilisera les informations ICMP pour mettre à jour ses tables de routage.

## 8.1.7 La couche transport : les protocoles UDP et TCP

### 8.1.7.1 Le protocole UDP : User Datagram Protocol

Le protocole UDP permet un échange non fiable de données entre deux points, c'est à dire que le bon acheminement d'un paquet à sa destination n'est pas garanti. L'application, si elle le souhaite peut gérer cela elle-même, en attendant par exemple après l'envoi d'un message, un accusé de réception, avant d'envoyer le suivant.

Pour l'instant, au niveau réseau, nous avons parlé d'adresses IP de machines. Or sur une machine, peuvent coexister en même temps différents processus qui tous peuvent communiquer. Il faut donc indiquer, lors de l'envoi d'un message, non seulement l'adresse IP de la machine destinatrice, mais également le "nom" du processus destinataire. Ce nom est en fait un numéro, appelé **numéro de port**. Certains numéros sont réservés à des applications standard : port 69 pour l'application **tftp** (*trivial file transfer protocol*) par exemple. Les paquets gérés par le protocole UDP sont appelés également des **datagrammes**. Ils ont la forme suivante :



Ces datagrammes seront encapsulés dans des paquets IP, puis dans des trames physiques.

### 8.1.7.2 Le protocole TCP : Transfer Control Protocol

Pour des communications sûres, le protocole UDP est insuffisant : le développeur d'applications doit élaborer lui-même un protocole lui permettant de détecter le bon acheminement des paquets.

Le protocole **TCP** (*Transfer Control Protocol*) évite ces problèmes. Ses caractéristiques sont les suivantes :

- Le processus qui souhaite émettre établit tout d'abord une **connexion** avec le processus destinataire des informations qu'il va émettre. Cette connexion se fait entre un port de la machine émettrice et un port de la machine réceptrice. Il y a entre les deux ports un chemin virtuel qui est ainsi créé et qui sera réservé aux deux seuls processus ayant réalisé la connexion. Tous les paquets émis par le processus source suivent ce chemin virtuel et arrivent dans l'ordre où ils ont été émis ce qui n'était pas garanti dans le protocole UDP puisque les paquets pouvaient suivre des chemins différents.
- L'information émise a un aspect continu. Le processus émetteur envoie des informations à son rythme. Celles-ci ne sont pas nécessairement envoyées tout de suite : le protocole TCP attend d'en avoir assez pour les envoyer. Elles sont stockées dans une structure appelée *segment TCP*. Ce segment une fois rempli sera transmis à la couche IP où il sera encapsulé dans un paquet IP.
- Chaque segment envoyé par le protocole TCP est numéroté. Le protocole TCP destinataire vérifie qu'il reçoit bien les segments en séquence. Pour chaque segment correctement reçu, il envoie un accusé de réception à l'expéditeur.
- Lorsque ce dernier le reçoit, il l'indique au processus émetteur. Celui-ci peut donc savoir qu'un segment est arrivé à bon port, ce qui n'était pas possible avec le protocole UDP.
- Si au bout d'un certain temps, le protocole TCP ayant émis un segment ne reçoit pas d'accusé de réception, il retransmet le segment en question, garantissant ainsi la qualité du service d'acheminement de l'information.
- Le circuit virtuel établi entre les deux processus qui communiquent est *full-duplex* : cela signifie que l'information peut transiter dans les deux sens. Ainsi le processus destination peut envoyer des accusés de réception alors même que le processus source continue d'envoyer des informations. Cela permet par exemple au protocole TCP source d'envoyer plusieurs segments sans attendre d'accusé de réception. S'il réalise au bout d'un certain temps qu'il n'a pas reçu l'accusé de réception d'un certain segment n° *n*, il reprendra l'émission des segments à ce point.

## 8.1.8 La couche Applications

Au-dessus des protocoles UDP et TCP, existent divers protocoles standard :

### TELNET

Ce protocole permet à un utilisateur d'une machine A du réseau de se connecter sur une machine B (appelée souvent machine hôte). TELNET émule sur la machine A un terminal dit universel. L'utilisateur se comporte donc comme s'il disposait d'un terminal connecté à la machine B. Telnet s'appuie sur le protocole TCP.

### FTP : (File Transfer protocol)

Ce protocole permet l'échange de fichiers entre deux machines distantes ainsi que des manipulations de fichiers tels que des créations de répertoire par exemple. Il s'appuie sur le protocole TCP.

### TFTP: (Trivial File Transfer Control)

Ce protocole est une variante de FTP. Il s'appuie sur le protocole UDP et est moins sophistiqué que FTP.

### DNS : (Domain Name System)

Lorsqu'un utilisateur désire échanger des fichiers avec une machine distante, par FTP par exemple, il doit connaître l'adresse Internet de cette machine. Par exemple, pour faire du FTP sur la machine Lagaffe de l'université d'Angers, il faudrait lancer FTP comme suit : FTP 193.49.144.1

Cela oblige à avoir un annuaire faisant la correspondance machine <--> adresse IP. Probablement que dans cet annuaire les machines seraient désignées par des noms symboliques tels que :

machine DPX2/320 de l'université d'Angers  
machine Sun de l'ISERPA d'Angers

On voit bien qu'il serait plus agréable de désigner une machine par un nom plutôt que par son adresse IP. Se pose alors le problème de l'unicité du nom : il y a des millions de machines interconnectées. On pourrait imaginer qu'un organisme centralisé attribue les noms. Ce serait sans doute assez lourd. Le contrôle des noms a été en fait distribué dans des **domaines**. Chaque domaine est géré par un organisme généralement très léger qui a toute liberté quant au choix des noms de machines. Ainsi les machines en France appartiennent au domaine **fr**, domaine géré par l'Inria de Paris. Pour continuer à simplifier les choses, on distribue encore le contrôle : des domaines sont créés à l'intérieur du domaine **fr**. Ainsi l'université d'Angers appartient au domaine **univ-Angers**. Le service gérant ce domaine a toute liberté pour nommer les machines du réseau de l'Université d'Angers. Pour l'instant ce domaine n'a pas été subdivisé. Mais dans une grande université comportant beaucoup de machines en réseau, il pourrait l'être.

La machine DPX2/320 de l'université d'Angers a été nommée *Lagaffe* alors qu'un PC 486DX50 a été nommé *liny*. Comment référencer ces machines de l'extérieur ? En précisant la hiérarchie des domaines auxquelles elles appartiennent. Ainsi le nom complet de la machine Lagaffe sera :

**Lagaffe.univ-Angers.fr**

A l'intérieur des domaines, on peut utiliser des noms relatifs. Ainsi à l'intérieur du domaine **fr** et en dehors du domaine **univ-Angers**, la machine Lagaffe pourra être référencée par

**Lagaffe.univ-Angers**

Enfin, à l'intérieur du domaine *univ-Angers*, elle pourra être référencée simplement par

**Lagaffe**

Une application peut donc référencer une machine par son nom. Au bout du compte, il faut quand même obtenir l'adresse Internet de cette machine. Comment cela est-il réalisé ? Supposons que d'une machine A, on veuille communiquer avec une machine B.

- si la machine B appartient au même domaine que la machine A, on trouvera probablement son adresse IP dans un fichier de la machine A.
- sinon, la machine A trouvera dans un autre fichier ou le même que précédemment, une liste de quelques **serveurs de noms** avec leurs adresses IP. Un *serveur de noms* est chargé de faire la correspondance entre un nom de machine et son adresse IP. La machine A va envoyer une requête spéciale au premier serveur de nom de sa liste, appelé requête DNS incluant donc le nom de la machine recherchée. Si le serveur interrogé a ce nom dans ses tablettes, il enverra à la machine A, l'adresse IP correspondante. Sinon, le serveur trouvera lui aussi dans ses fichiers, une liste de serveurs de noms qu'il peut interroger. Il le fera alors. Ainsi un certain nombre de serveurs de noms vont être interrogés, pas de façon anarchique mais d'une façon à minimiser les requêtes. Si la machine est finalement trouvée, la réponse redescendra jusqu'à la machine A.

### **XDR : (eXternal Data Representation)**

Créé par sun Microsystems, ce protocole spécifie une représentation standard des données, indépendante des machines.

### **RPC : (Remote Procedure Call)**

Défini également par sun, c'est un protocole de communication entre applications distantes, indépendant de la couche transport. Ce protocole est important : il décharge le programmeur de la connaissance des détails de la couche transport et rend les applications portables. Ce protocole s'appuie sur le protocole XDR

### **NFS : Network File System**

Toujours défini par Sun, ce protocole permet à une machine, de "voir" le système de fichiers d'une autre machine. Il s'appuie sur le protocole RPC précédent.

## 8.1.9 Conclusion

Nous avons présenté dans cette introduction quelques grandes lignes des protocoles Internet. Pour approfondir ce domaine, on pourra lire l'excellent livre de Douglas Comer :

Titre TCP/IP : Architecture, Protocoles, Applications.  
Auteur Douglas COMER

## 8.2 Gestion des adresses réseau

Une machine sur le réseau Internet est définie de façon unique par une adresse IP (Internet Protocol) de la forme I1.I2.I3.I4 où I<sub>n</sub> est un nombre entre 1 et 254. Elle peut être également définie par un nom également unique. Ce nom n'est pas obligatoire, les applications utilisant toujours au final les adresses IP des machines. Ils sont là pour faciliter la vie des utilisateurs. Ainsi il est plus facile, avec un navigateur, de demander l'URL *http://www.ibm.com* que l'URL *http://129.42.17.99* bien que les deux méthodes soient possibles. L'association adresse IP <--> nomMachine est assurée par un service distribué de l'internet appelé DNS (Domain Name System). La plate-forme .NET offre la classe **Dns** pour gérer les adresses internet :

Bibliothèque de classes .NET Framework

**Dns, classe** [Visual Basic]

Fournit des fonctionnalités de résolution de noms de domaines simples.

Pour obtenir une liste de tous les membres de ce type, consultez [Dns, membres](#).

[System.Object](#)  
**System.Net.Dns**

**NotInheritable Public Class Dns**

La plupart des méthodes de la classe sont statiques. Regardons celles qui nous intéressent :

```
Overloads Public Shared Function
GetHostByAddress(ByVal address As
String) As IPHostEntry

Public Shared Function
GetHostByName(ByVal hostName As
String) As IPHostEntry

Public Shared Function
GetHostName() As String
```

rend une adresse *IPHostEntry* à partir d'une adresse IP sous la forme "I1.I2.I3.I4". Lance une exception si la machine *address* ne peut être trouvée.

rend une adresse *IPHostEntry* à partir d'un nom de machine. Lance une exception si la machine *name* ne peut être trouvée.

rend le nom de la machine sur laquelle s'exécute le programme qui joue cette instruction

Les adresses réseau de type *IPHostEntry* ont la forme suivante :

Bibliothèque de classes .NET Framework

**IPHostEntry, membres**

[IPHostEntry, vue d'ensemble](#)

**Constructeurs publics**

<a href="#">IPHostEntry, constructeur</a> Pris en charge par le .NET Compact Framework.	Initialise une nouvelle instance de la classe <b>IPHostEntry</b> .
--	--

**Propriétés publiques**

<a href="#">AddressList</a> Pris en charge par le .NET Compact Framework.	Obtient ou définit une liste d'adresses IP associées à un hôte.
<a href="#">Aliases</a> Pris en charge par le .NET Compact Framework.	Obtient ou définit une liste d'alias associées à un hôte.
<a href="#">HostName</a> Pris en charge par le .NET Compact Framework.	Obtient ou définit le nom DNS de l'hôte.

Les propriétés qui nous intéressent :

Public Property AddressList As  
IPAddress ()

liste des adresses IP d'une machine. Si une adresse IP désigne une et une seule machine physique, une machine physique peut elle avoir plusieurs adresses IP. Ce sera le cas si elle a plusieurs cartes réseau qui la connectent à des réseaux différents.

Public Property Aliases As  
String ()

liste des alias d'une machine, pouvant être désignée par un nom principal et des alias

Public Property HostName As  
String

le nom de la machine si elle en a un

De la classe *IPAddress* nous retiendrons le constructeur, les propriétés et méthodes suivantes :

## IPAddress, classe [Visual Basic]

Fournit une adresse IP (Internet Protocol).

Pour obtenir une liste de tous les membres de ce type, consultez [IPAddress, membres](#).

[System.Object](#)

**System.Net.IPAddress**

```
<Serializable>  
Public Class IPAddress
```

Un objet [IPAddress] peut être transformé en chaîne *11.12.13.14* avec la méthode *ToString()*. Inversement, on peut obtenir un objet *IPAddress* à partir d'une chaîne *11.12.13.14* avec la méthode statique *IPAddress.Parse("11.12.13.14")*. Considérons le programme suivant qui affiche le nom de la machine sur laquelle il s'exécute puis de façon interactive donne les correspondances adresse IP <--> nom Machine :

```
dos>address1  
Machine Locale=tahe  
  
Machine recherchée (fin pour arrêter) : istia.univ-angers.fr  
Machine : istia.univ-angers.fr  
Adresses IP : 193.49.146.171  
  
Machine recherchée (fin pour arrêter) : 193.49.146.171  
Machine : istia.istia.univ-angers.fr  
Adresses IP : 193.49.146.171  
Alias : 171.146.49.193.in-addr.arpa  
  
Machine recherchée (fin pour arrêter) : www.ibm.com  
Machine : www.ibm.com  
Adresses IP : 129.42.17.99,129.42.18.99,129.42.19.99,129.42.16.99  
  
Machine recherchée (fin pour arrêter) : 129.42.17.99  
Machine : www.ibm.com  
Adresses IP : 129.42.17.99  
  
Machine recherchée (fin pour arrêter) : x.y.z  
Impossible de trouver la machine [x.y.z]  
  
Machine recherchée (fin pour arrêter) : localhost  
Machine : tahe  
Adresses IP : 127.0.0.1  
  
Machine recherchée (fin pour arrêter) : 127.0.0.1  
Machine : tahe  
Adresses IP : 127.0.0.1  
  
Machine recherchée (fin pour arrêter) : tahe  
Machine : tahe  
Adresses IP : 127.0.0.1  
  
Machine recherchée (fin pour arrêter) : fin
```

Le programme est le suivant :

```
' options  
Option Explicit On  
Option Strict On  
  
' espaces de noms
```

```

Imports System
Imports System.Net
Imports System.Text.RegularExpressions

' module de test
Public Module addresses

Sub Main()
' affiche le nom de la machine locale
' puis donne interactivement des infos sur les machines réseau
' identifiées par un nom ou une adresse IP
' machine locale
Dim localHost As String = Dns.GetHostName()
Console.Out.WriteLine("Machine Locale=" + localHost)

' question-réponses interactives
Dim machine As String
Dim adresseMachine As IPHostEntry
While True
' saisie du nom de la machine recherchée
Console.Out.Write("Machine recherchée (fin pour arrêter) : ")
machine = Console.In.ReadLine().Trim().ToLower()
' fini ?
If machine = "fin" Then
Exit While
End If

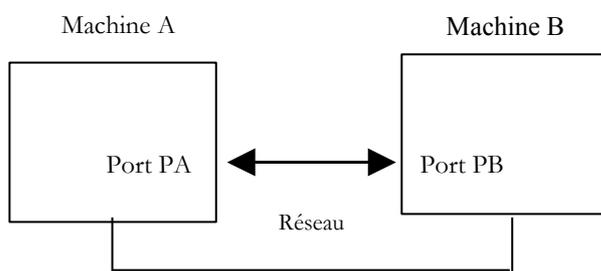
' adresse I1.I2.I3.I4 ou nom de machine ?
Dim isIPv4 As Boolean = Regex.IsMatch(machine, "^\s*\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\s*$")
' gestion exception
Try
If isIPv4 Then
adresseMachine = Dns.GetHostByAddress(machine)
Else
adresseMachine = Dns.GetHostByName(machine)
End If
' le nom
Console.Out.WriteLine("Machine : " + adresseMachine.HostName)
' les adresses IP
Console.Out.Write("Adresses IP : " + adresseMachine.AddressList(0).ToString())
Dim i As Integer
For i = 1 To adresseMachine.AddressList.Length - 1
Console.Out.Write(", " + adresseMachine.AddressList(i).ToString())
Next i
Console.Out.WriteLine()
' les alias
If adresseMachine.Aliases.Length <> 0 Then
Console.Out.Write("Alias : " + adresseMachine.Aliases(0))
For i = 1 To adresseMachine.Aliases.Length - 1
Console.Out.Write(", " + adresseMachine.Aliases(i))
Next i
Console.Out.WriteLine()
End If
Catch
' la machine n'existe pas
Console.Out.WriteLine("Impossible de trouver la machine [" + machine + "]")
End Try
End While
End Sub
End Module

```

## 8.3 Programmation TCP-IP

### 8.3.1 Généralités

Considérons la communication entre deux machines distantes A et B :



Lorsque une application *AppA* d'une machine A veut communiquer avec une application *AppB* d'une machine B de l'Internet, elle doit connaître plusieurs choses :

- l'adresse IP ou le nom de la machine B
- le numéro du port avec lequel travaille l'application *AppB*. En effet la machine B peut supporter de nombreuses applications qui travaillent sur l'Internet. Lorsqu'elle reçoit des informations provenant du réseau, elle doit savoir à quelle application sont destinées ces informations. Les applications de la machine B ont accès au réseau via des guichets appelés également des ports de communication. Cette information est contenue dans le paquet reçu par la machine B afin qu'il soit délivré à la bonne application.
- les protocoles de communication compris par la machine B. Dans notre étude, nous utiliserons uniquement les protocoles TCP-IP.
- le protocole de dialogue accepté par l'application *AppB*. En effet, les machines A et B vont se "parler". Ce qu'elles vont dire va être encapsulé dans les protocoles TCP-IP. Néanmoins, lorsqu'au bout de la chaîne, l'application *AppB* va recevoir l'information envoyée par l'application *AppA*, il faut qu'elle soit capable de l'interpréter. Ceci est analogue à la situation où deux personnes A et B communiquent par téléphone : leur dialogue est transporté par le téléphone. La parole va être codée sous forme de signaux par le téléphone A, transportée par des lignes téléphoniques, arriver au téléphone B pour y être décodée. La personne B entend alors des paroles. C'est là qu'intervient la notion de protocole de dialogue : si A parle français et que B ne comprend pas cette langue, A et B ne pourront dialoguer utilement.

Aussi les deux applications communicantes doivent -elles être d'accord sur le type de dialogue qu'elles vont adopter. Par exemple, le dialogue avec un service *ftp* n'est pas le même qu'avec un service *pop* : ces deux services n'acceptent pas les mêmes commandes. Elles ont un protocole de dialogue différent.

## 8.3.2 Les caractéristiques du protocole TCP

Nous n'étudierons ici que des communications réseau utilisant le protocole de transport TCP. Rappelons ici, les caractéristiques de celui-ci :

- Le processus qui souhaite émettre établit tout d'abord une **connexion** avec le processus destinataire des informations qu'il va émettre. Cette connexion se fait entre un port de la machine émettrice et un port de la machine réceptrice. Il y a entre les deux ports un chemin virtuel qui est ainsi créé et qui sera réservé aux deux seuls processus ayant réalisé la connexion.
- Tous les paquets émis par le processus source suivent ce chemin virtuel et arrivent dans l'ordre où ils ont été émis
- L'information émise a un aspect continu. Le processus émetteur envoie des informations à son rythme. Celles-ci ne sont pas nécessairement envoyées tout de suite : le protocole TCP attend d'en avoir assez pour les envoyer. Elles sont stockées dans une structure appelée *segment TCP*. Ce segment une fois rempli sera transmis à la couche IP où il sera encapsulé dans un paquet IP.
- Chaque segment envoyé par le protocole TCP est numéroté. Le protocole TCP destinataire vérifie qu'il reçoit bien les segments en séquence. Pour chaque segment correctement reçu, il envoie un accusé de réception à l'expéditeur.
- Lorsque ce dernier le reçoit, il l'indique au processus émetteur. Celui-ci peut donc savoir qu'un segment est arrivé à bon port.
- Si au bout d'un certain temps, le protocole TCP ayant émis un segment ne reçoit pas d'accusé de réception, il retransmet le segment en question, garantissant ainsi la qualité du service d'acheminement de l'information.
- Le circuit virtuel établi entre les deux processus qui communiquent est *full-duplex* : cela signifie que l'information peut transiter dans les deux sens. Ainsi le processus destination peut envoyer des accusés de réception alors même que le processus source continue d'envoyer des informations. Cela permet par exemple au protocole TCP source d'envoyer plusieurs segments sans attendre d'accusé de réception. S'il réalise au bout d'un certain temps qu'il n'a pas reçu l'accusé de réception d'un certain segment n° *n*, il reprendra l'émission des segments à ce point.

## 8.3.3 La relation client-serveur

Souvent, la communication sur Internet est dissymétrique : la machine A initie une connexion pour demander un service à la machine B : il précise qu'il veut ouvrir une connexion avec le service SB1 de la machine B. Celle-ci accepte ou refuse. Si elle accepte, la machine A peut envoyer ses demandes au service SB1. Celles-ci doivent se conformer au protocole de dialogue compris par le service SB1. Un dialogue demande-réponse s'instaure ainsi entre la machine A qu'on appelle machine **cliente** et la machine B qu'on appelle machine **serveur**. L'un des deux partenaires fermera la connexion.

## 8.3.4 Architecture d'un client

L'architecture d'un programme réseau demandant les services d'une application serveur sera la suivante :

```
ouvrir la connexion avec le service SB1 de la machine B
si réussite alors
  tant que ce n'est pas fini
    préparer une demande
    l'émettre vers la machine B
```

```
    attendre et récupérer la réponse
    la traiter
  fin tant que
finsi
```

### 8.3.5 Architecture d'un serveur

L'architecture d'un programme offrant des services sera la suivante :

```
ouvrir le service sur la machine locale
tant que le service est ouvert
  se mettre à l'écoute des demandes de connexion sur un port dit port d'écoute
  lorsqu'il y a une demande, la faire traiter par une autre tâche sur un autre port dit port de service
fin tant que
```

Le programme serveur traite différemment la demande de connexion initiale d'un client de ses demandes ultérieures visant à obtenir un service. Le programme n'assure pas le service lui-même. S'il le faisait, pendant la durée du service il ne serait plus à l'écoute des demandes de connexion et des clients ne seraient alors pas servis. Il procède donc autrement : dès qu'une demande de connexion est reçue sur le port d'écoute puis acceptée, le serveur crée une tâche chargée de rendre le service demandé par le client. Ce service est rendu sur un autre port de la machine serveur appelé **port de service**. On peut ainsi servir plusieurs clients en même temps. Une tâche de service aura la structure suivante :

```
tant que le service n'a pas été rendu totalement
  attendre une demande sur le port de service
  lorsqu'il y en a une, élaborer la réponse
  transmettre la réponse via le port de service
fin tant que
  libérer le port de service
```

### 8.3.6 La classe TcpClient

La classe **TcpClient** est la classe qui convient pour représenter le client d'un service TCP. Elle est définie comme suit :

[System.Object](#)

**System.Net.Sockets.TcpClient**

```
Public Class TcpClient
Implements IDisposable
```

Les constructeurs, méthodes et propriétés qui nous intéressent sont les suivants :

```
Public Sub New(ByVal hostname
As String,ByVal port As
Integer)
```

crée une liaison tcp avec le serveur opérant sur le port indiqué (*port*) de la machine indiquée (*hostname*). Par exemple `new TcpClient("istia.univ-angers.fr",80)` pour se connecter au port 80 de la machine *istia.univ-angers.fr*

```
Public Sub Close()
```

ferme la connexion au serveur Tcp

```
Public Function GetStream() As
NetworkStream
```

obtient un flux **NetworkStream** de lecture et d'écriture vers le serveur. C'est ce flux qui permet les échanges client-serveur.

### 8.3.7 La classe NetworkStream

La classe **NetworkStream** représente le flux réseau entre le client et le serveur. La classe est définie comme suit :

[System.Object](#)

[System.MarshalByRefObject](#)

[System.IO.Stream](#)

**System.Net.Sockets.NetworkStream**

```
Public Class NetworkStream
Inherits Stream
```

La classe *NetworkStream* est dérivée de la classe *Stream*. Beaucoup d'applications client-serveur échangent des lignes de texte terminées par les caractères de fin de ligne "\r\n". Aussi il est intéressant d'utiliser des objets *StreamReader* et *StreamWriter* pour lire et écrire ces lignes dans le flux réseau. Lorsque deux machines communiquent, il y a à chaque bout de la liaison un objet *TcpClient*.

La méthode *GetStream* de cet objet permet d'avoir accès au flux réseau (*NetworkStream*) qui lie les deux machines. Ainsi si une machine M1 a établi une liaison avec une machine M2 à l'aide d'un objet *TcpClient client1* qu'elles échangent des lignes de texte, elle pourra créer ses flux de lecture et écriture de la façon suivante :

```
Dim in1 as StreamReader=new StreamReader(client1.GetStream())
Dim out1 as StreamWriter=new StreamWriter(client1.GetStream())
out1.AutoFlush=true
```

L'instruction

```
out1.AutoFlush=true
```

signifie que le flux d'écriture de *client1* ne transitera pas par un buffer intermédiaire mais ira directement sur le réseau. Ce point est important. En général lorsque *client1* envoie une ligne de texte à son partenaire il en attend une réponse. Celle-ci ne viendra jamais si la ligne a été en réalité bufferisée sur la machine M1 et jamais envoyée. Pour envoyer une ligne de texte à la machine M2, on écrira :

```
client1.WriteLine("un texte")
```

Pour lire la réponse de M2, on écrira :

```
Dim réponse as String=client1.ReadLine()
```

### 8.3.8 Architecture de base d'un client internet

Nous avons maintenant les éléments pour écrire l'architecture de base d'un client internet :

```
Dim client As TcpClient = Nothing ' le client
Dim [IN] As StreamReader = Nothing ' le flux de lecture du client
Dim OUT As StreamWriter = Nothing ' le flux d'écriture du client
Dim demande As String = Nothing ' demande du client
Dim réponse As String = Nothing ' réponse du serveur

Try
    ' on se connecte au service officiant sur le port P de la machine M
    client = New TcpClient(nomServeur, port)

    ' on crée les flux d'entrée-sortie du client TCP
    [IN] = New StreamReader(client.GetStream())
    OUT = New StreamWriter(client.GetStream())
    OUT.AutoFlush = True

    ' boucle demande - réponse
    While True
        ' on prépare la demande
        demande = ...
        ' on l'envoie au serveur
        OUT.WriteLine(demande)
        ' on lit la réponse du serveur
        réponse = [IN].ReadLine()
        ' on traite la réponse
        ...
    End While
    ' c'est fini
    client.Close()
Catch ex As Exception
    ' on gère l'exception
    ...
End Try
```

### 8.3.9 La classe *TcpListener*

La classe ***TcpListener*** est la classe qui convient pour représenter un service TCP. Elle est définie comme suit :

[System.Object](#)

**System.Net.Sockets.TcpListener**

```
Public Class TcpListener
```

Les constructeurs, méthodes et propriétés qui nous intéressent sont les suivants :

```
Public Sub New(ByVal localaddr As IPAddress,ByVal port As Integer)
```

crée un service TCP qui va attendre (*listen*) les demandes des clients sur un port passé en paramètre (*port*) appelé port d'écoute de la machine locale d'adresse IP localadr.

```
Public Function AcceptTcpClient()  
As TcpClient
```

accepte la demande d'un client. Rend comme résultat un objet *TcpClient* associé à un autre port, appelé port de service.

```
Public Sub Start()
```

lance l'écoute des demandes clients

```
Public Sub Stop()
```

arrête d'écouter les demandes clients

### 8.3.10 Architecture de base d'un serveur Internet

De ce qui a été vu précédemment, on peut déduire la structure de base d'un serveur :

```
' on crée le servive d'écoute  
Dim ecoute As TcpListener = Nothing  
Dim port As Integer = ...  
Try  
    ' on crée le service  
    ecoute = New TcpListener(IPAddress.Parse("127.0.0.1"), port)  
    ' on le lance  
    ecoute.Start()  
    ' boucle de service  
    Dim liaisonClient As TcpClient = Nothing  
    While not fini  
        ' attente d'un client  
        liaisonClient = ecoute.AcceptTcpClient()  
        ' le service est assuré par une autre tâche  
        Dim tache As Thread = New Thread(New ThreadStart(AddressOf [méthode]))  
        tache.Start()  
    End While  
Catch ex As Exception  
    ' on signale l'erreur  
....  
End Try  
' fin du service  
ecoute.Stop()
```

La classe *Service* est un *thread* qui pourrait avoir l'allure suivante :

```
Public Class Service  
  
    Private liaisonClient As TcpClient ' liaison avec le client  
    Private [IN] As StreamReader ' flux d'entrée  
    Private OUT As StreamWriter ' flux de sortie  
  
    ' constructeur  
    Public Sub New(ByVal liaisonClient As TcpClient, ...)  
        Me.liaisonClient = liaisonClient  
        ...  
    End Sub  
  
    ' méthode run  
    Public Sub Run()  
        ' rend le service au client  
        Try  
            ' flux d'entrée  
            [IN] = New StreamReader(liaisonClient.GetStream())  
            ' flux de sortie  
            OUT = New StreamWriter(liaisonClient.GetStream())  
            OUT.AutoFlush = True  
            ' boucle lecture demande/écriture réponse  
            Dim demande As String = Nothing  
            Dim reponse As String = Nothing  
            demande = [IN].ReadLine  
            While Not (demande Is Nothing)  
                ' on traite la demande  
                ...  
                ' on envoie la réponse  
                reponse = "[" + demande + "]"  
                OUT.WriteLine(reponse)  
                ' demande suivante  
                demande = [IN].ReadLine  
            End While  
            ' fin liaison  
            liaisonClient.Close()  
        Catch e As Exception  
            ...  
        End Try  
        ' fin du service  
    End Sub
```

## 8.4 Exemples

### 8.4.1 Serveur d'écho

Nous nous proposons d'écrire un serveur d'écho qui sera lancé depuis une fenêtre DOS par la commande :

**serveurEcho port**

Le serveur officie sur le port passé en paramètre. Il se contente de renvoyer au client la demande que celui-ci lui a envoyée. Le programme est le suivant :

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System.Net.Sockets
Imports System.Net
Imports System
Imports System.IO
Imports System.Threading
Imports Microsoft.VisualBasic

' appel : serveurEcho port
' serveur d'écho
' renvoie au client la ligne que celui-ci lui a envoyée

Public Class serveurEcho
    Private Shared syntaxe As String = "Syntaxe : serveurEcho port"

    ' programme principal
    Public Shared Sub Main(ByVal args() As String)

        ' y-a-t-il un argument
        If args.Length <> 1 Then
            erreur(syntaxe, 1)
        End If

        ' cet argument doit être entier >0
        Dim port As Integer = 0
        Dim erreurPort As Boolean = False
        Dim E As Exception = Nothing
        Try
            port = Integer.Parse(args(0))
        Catch ex As Exception
            E = ex
            erreurPort = True
        End Try
        erreurPort = erreurPort Or port <= 0
        If erreurPort Then
            erreur(syntaxe + ControlChars.Lf + "Port incorrect (" + E.ToString + ")", 2)
        End If

        ' on crée le servive d'écoute
        Dim ecoute As TcpListener = Nothing
        Dim nbClients As Integer = 0 ' nbre de clients traités
        Try
            ' on crée le service
            ecoute = New TcpListener(IPAddress.Parse("127.0.0.1"), port)
            ' on le lance
            ecoute.Start()
            ' suivi
            Console.Out.WriteLine("Serveur d'écho lancé sur le port " & port)
            Console.Out.WriteLine(ecoute.LocalEndpoint)

            ' boucle de service
            Dim liaisonClient As TcpClient = Nothing
            While True
                ' boucle infinie - sera arrêtée par Ctrl-C
                ' attente d'un client
                liaisonClient = ecoute.AcceptTcpClient()

                ' le service est assuré par une autre tâche
                nbClients += 1
                Dim tache As Thread = New Thread(New ThreadStart(AddressOf New traiteClientEcho(liaisonClient, nbClients).Run))
                tache.Start()
            End While
            ' on retourne à l'écoute des demandes
        End Try
    End Sub
End Class
```

```

Catch ex As Exception
    ' on signale l'erreur
    erreur("L'erreur suivante s'est produite : " + ex.Message, 3)
End Try
' fin du service
ecoute.Stop()
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

' -----
' assure le service à un client du serveur d'écho
Public Class traiteClientEcho

    Private liaisonClient As TcpClient ' liaison avec le client
    Private numClient As Integer ' n° de client
    Private [IN] As StreamReader ' flux d'entrée
    Private OUT As StreamWriter ' flux de sortie

    ' constructeur
    Public Sub New(ByVal liaisonClient As TcpClient, ByVal numClient As Integer)
        Me.liaisonClient = liaisonClient
        Me.numClient = numClient
    End Sub

    ' méthode run
    Public Sub Run()
        ' rend le service au client
        Console.Out.WriteLine("Début de service au client " & numClient)
        Try
            ' flux d'entrée
            [IN] = New StreamReader(liaisonClient.GetStream())
            ' flux de sortie
            OUT = New StreamWriter(liaisonClient.GetStream())
            OUT.AutoFlush = True
            ' boucle lecture demande/écriture réponse
            Dim demande As String = Nothing
            Dim reponse As String = Nothing
            demande = [IN].ReadLine
            While Not (demande Is Nothing)
                ' suivi
                Console.Out.WriteLine(("Client " & numClient & " : " & demande))
                ' le service s'arrête lorsque le client envoie une marque de fin de fichier
                reponse = "[" + demande + "]"
                OUT.WriteLine(reponse)
                ' le service s'arrête lorsque le client envoie "fin"
                If demande.Trim().ToLower() = "fin" Then
                    Exit While
                End If
                ' demande suivante
                demande = [IN].ReadLine
            End While
            ' fin liaison
            liaisonClient.Close()
        Catch e As Exception
            erreur("Erreur lors de la fermeture de la liaison client (" + e.ToString + ")", 2)
        End Try
        ' fin du service
        Console.Out.WriteLine("Fin de service au client " & numClient)
    End Sub

    ' affichage des erreurs
    Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
        ' affichage erreur
        System.Console.Error.WriteLine(msg)
        ' arrêt avec erreur
        Environment.Exit(exitCode)
    End Sub
End Class

```

La structure du serveur est conforme à l'architecture générale des serveurs tcp.

## 8.4.2 Un client pour le serveur d'écho

Nous écrivons maintenant un client pour le serveur précédent. Il sera appelé de la façon suivante :

**clientEcho nomServeur port**

Il se connecte à la machine *nomServeur* sur le port *port* puis envoie au serveur des lignes de texte que celui-ci lui renvoie en écho.

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports System.Net.Sockets
Imports System.Net
Imports System
Imports System.IO
Imports System.Threading
Imports Microsoft.VisualBasic

Public Class clientEcho

    ' se connecte à un serveur d'écho
    ' toute ligne tapée au clavier est alors reçue en écho
    Public Shared Sub Main(ByVal args() As String)
        ' syntaxe
        Const syntaxe As String = "pg machine port"

        ' nombre d'arguments
        If args.Length <> 2 Then
            erreur(syntaxe, 1)
        End If
        ' on note le nom du serveur
        Dim nomServeur As String = args(0)

        ' le port doit être entier >0
        Dim port As Integer = 0
        Dim erreurPort As Boolean = False
        Dim E As Exception = Nothing
        Try
            port = Integer.Parse(args(1))
        Catch ex As Exception
            E = ex
            erreurPort = True
        End Try
        erreurPort = erreurPort Or port <= 0
        If erreurPort Then
            erreur(syntaxe + ControlChars.Lf + "Port incorrect (" + E.ToString + ")", 2)
        End If

        ' on peut travailler
        Dim client As TcpClient = Nothing ' le client
        Dim [IN] As StreamReader = Nothing ' le flux de lecture du client
        Dim OUT As StreamWriter = Nothing ' le flux d'écriture du client
        Dim demande As String = Nothing ' demande du client
        Dim réponse As String = Nothing ' réponse du serveur
        Try
            ' on se connecte au service officiant sur le port P de la machine M
            client = New TcpClient(nomServeur, port)

            ' on crée les flux d'entrée-sortie du client TCP
            [IN] = New StreamReader(client.GetStream())
            OUT = New StreamWriter(client.GetStream())
            OUT.AutoFlush = True

            ' boucle demande - réponse
            While True
                ' la demande vient du clavier
                Console.Out.Write("demande (fin pour arrêter) : ")
                demande = Console.In.ReadLine()
                ' on l'envoie au serveur
                OUT.WriteLine(demande)
                ' on lit la réponse du serveur
                réponse = [IN].ReadLine()
                ' on traite la réponse
                Console.Out.WriteLine(("Réponse : " + réponse))
                ' fini ?
                If demande.Trim().ToLower() = "fin" Then

```

```

        Exit While
    End If
End While
' c'est fini
client.Close()
Catch ex As Exception
    ' on gère l'exception
    erreur(ex.Message, 3)
End Try
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

```

La structure de ce client est conforme à l'architecture générale des clients *tcp*. Voici les résultats obtenus dans la configuration suivante :

- le serveur est lancé sur le port 100 dans une fenêtre Dos
- sur la même machine deux clients sont lancés dans deux autres fenêtres Dos

Dans la fenêtre du client 1 on a les résultats suivants :

```

dos>clientEcho localhost 100
demande (fin pour arrêter) : ligne1
Réponse : [ligne1]
demande (fin pour arrêter) : ligne1B
Réponse : [ligne1B]
demande (fin pour arrêter) : ligne1C
Réponse : [ligne1C]
demande (fin pour arrêter) : fin
Réponse : [fin]

```

Dans celle du client 2 :

```

dos>clientEcho localhost 100
demande (fin pour arrêter) : ligne2A
Réponse : [ligne2A]
demande (fin pour arrêter) : ligne2B
Réponse : [ligne2B]
demande (fin pour arrêter) : fin
Réponse : [fin]

```

Dans celle du serveur :

```

dos>serveurEcho 100
Serveur d'écho lancé sur le port 100
0.0.0.0:100
Début de service au client 1
Client 1 : ligne1
Début de service au client 2
Client 2 : ligne2A
Client 2 : ligne2B
Client 1 : ligne1B
Client 1 : ligne1C
Client 2 : fin
Fin de service au client 2
Client 1 : fin
Fin de service au client 1
^C

```

On remarquera que le serveur a bien été capable de servir deux clients simultanément.

### 8.4.3 Un client TCP générique

Beaucoup de services créés à l'origine de l'Internet fonctionnent selon le modèle du serveur d'écho étudié précédemment : les échanges client-serveur se font pas échanges de lignes de texte. Nous allons écrire un client *tcp* générique qui sera lancé de la façon suivante : **cltgen serveur port**

Ce client TCP se connectera sur le port *port* du serveur *serveur*. Ceci fait, il créera deux threads :

1. un thread chargé de lire des commandes tapées au clavier et de les envoyer au serveur

- un thread chargé de lire les réponses du serveur et de les afficher à l'écran

Pourquoi deux threads alors que dans l'application précédente ce besoin ne s'était pas fait ressentir ? Dans cette dernière, le protocole du dialogue était connu : le client envoyait une seule ligne et le serveur répondait par une seule ligne. Chaque service a son protocole particulier et on trouve également les situations suivantes :

- le client doit envoyer plusieurs lignes de texte avant d'avoir une réponse
- la réponse d'un serveur peut comporter plusieurs lignes de texte

Aussi la boucle envoi d'une unique ligne au serveur - réception d'une unique ligne envoyée par le serveur ne convient-elle pas toujours. On va donc créer deux boucles dissociées :

- une boucle de lecture des commandes tapées au clavier pour être envoyées au serveur. L'utilisateur signalera la fin des commandes avec le mot clé *fin*.
- une boucle de réception et d'affichage des réponses du serveur. Celle-ci sera une boucle infinie qui ne sera interrompue que par la fermeture du flux réseau par le serveur ou par l'utilisateur au clavier qui tapera la commande *fin*.

Pour avoir ces deux boucles dissociées, il nous faut deux threads indépendants. Montrons un exemple d'exécution où notre client tcp générique se connecte à un service SMTP (SendMail Transfer Protocol). Ce service est responsable de l'acheminement du courrier électronique aux destinataires. Il fonctionne sur le port 25 et a un protocole de dialogue de type échanges de lignes de texte.

```
dos>cltgen istia.univ-angers.fr 25
Commandes :
<-- 220 istia.univ-angers.fr ESMTTP Sendmail 8.11.6/8.9.3; Mon, 13 May 2002 08:37:26 +0200
help
<-- 502 5.3.0 Sendmail 8.11.6 -- HELP not implemented
mail from: machin@univ-angers.fr
<-- 250 2.1.0 machin@univ-angers.fr... Sender ok
rcpt to: serge.tahe@istia.univ-angers.fr
<-- 250 2.1.5 serge.tahe@istia.univ-angers.fr... Recipient ok
data
<-- 354 Enter mail, end with "." on a line by itself
Subject: test

ligne1
ligne2
ligne3
.
<-- 250 2.0.0 g4D6bks25951 Message accepted for delivery
quit
<-- 221 2.0.0 istia.univ-angers.fr closing connection
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]
```

Commentons ces échanges client-serveur :

- le service SMTP envoie un message de bienvenue lorsqu'un client se connecte à lui :

```
<-- 220 istia.univ-angers.fr ESMTTP Sendmail 8.11.6/8.9.3; Mon, 13 May 2002 08:37:26 +0200
```

- certaines services ont une commande *help* donnant des indications sur les commandes utilisables avec le service. Ici ce n'est pas le cas. Les commandes SMTP utilisées dans l'exemple sont les suivantes :
  - mail from:** *expéditeur*, pour indiquer l'adresse électronique de l'expéditeur du message
  - rcpt to:** *destinataire*, pour indiquer l'adresse électronique du destinataire du message. S'il y a plusieurs destinataires, on ré-émet autant de fois que nécessaire la commande **rcpt to:** pour chacun des destinataires.
  - data** qui signale au serveur SMTP qu'on va envoyer le message. Comme indiqué dans la réponse du serveur, celui-ci est une suite de lignes terminée par une ligne contenant le seul caractère point. Un message peut avoir des entêtes séparés du corps du message par une ligne vide. Dans notre exemple, nous avons mis un sujet avec le mot clé **Subject:**
- une fois le message envoyé, on peut indiquer au serveur qu'on a terminé avec la commande **quit**. Le serveur ferme alors la connexion réseau. Le thread de lecture peut détecter cet événement et s'arrêter.
- l'utilisateur tape alors **fin** au clavier pour arrêter également le thread de lecture des commandes tapées au clavier.

Si on vérifie le courrier reçu, nous avons la chose suivante (Outlook) :

**From:** machin@univ-angers.fr **To:**  
**Subject:** test

ligne1  
ligne2  
ligne3

On remarquera que le service SMTP ne peut détecter si un expéditeur est valide ou non. Aussi ne peut-on jamais faire confiance au champ *from* d'un message. Ici l'expéditeur *machin@univ-angers.fr* n'existait pas. Ce client tcp générique peut nous permettre de découvrir le protocole de dialogue de services internet et à partir de là construire des classes spécialisées pour des clients de ces services. Découvrons le protocole de dialogue du service POP (Post Office Protocol) qui permet de retrouver ses méls stockés sur un serveur. Il travaille sur le port 110.

```
dos>cltgen istia.univ-angers.fr 110
Commandes :
<-- +OK Qpopper (version 4.0.3) at istia.univ-angers.fr starting.
help
<-- -ERR Unknown command: "help".
user st
<-- +OK Password required for st.
pass monpassword
<-- +OK st has 157 visible messages (0 hidden) in 11755927 octets.
list
<-- +OK 157 visible messages (11755927 octets)
<-- 1 892847
<-- 2 171661
...
<-- 156 2843
<-- 157 2796
<-- .
retr 157
<-- +OK 2796 octets
<-- Received: from lagaffe.univ-angers.fr (lagaffe.univ-angers.fr [193.49.144.1])
<--   by istia.univ-angers.fr (8.11.6/8.9.3) with ESMTP id g4D6wZs26600;
<--   Mon, 13 May 2002 08:58:35 +0200
<-- Received: from jaume ([193.49.146.242])
<--   by lagaffe.univ-angers.fr (8.11.1/8.11.2/Ge020000215) with SMTP id g4D6wSd37691;
<--   Mon, 13 May 2002 08:58:28 +0200 (CEST)
...
<-- -----
<-- NOC-RENATER2                Tl. : 0800 77 47 95
<-- Fax : (+33) 01 40 78 64 00 , Email : noc-r2@cssi.renater.fr
<-- -----
<-- .
quit
<-- +OK Pop server at istia.univ-angers.fr signing off.
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]
```

Les principales commandes sont les suivantes :

- **user** *login*, où on donne son login sur la machine qui détient nos méls
- **pass** *password*, où on donne le mot de passe associé au login précédent
- **list**, pour avoir la liste des messages sous la forme numéro, taille en octets
- **retr** *i*, pour lire le message n° *i*
- **quit**, pour arrêter le dialogue.

Découvrons maintenant le protocole de dialogue entre un client et un serveur Web qui lui travaille habituellement sur le port 80 :

```
dos>cltgen istia.univ-angers.fr 80
Commandes :
GET /index.html HTTP/1.0

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
```

```

<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>
<--
<-- <head>
<-- <meta http-equiv="Content-Type"
<-- content="text/html; charset=iso-8859-1">
<-- <meta name="GENERATOR" content="Microsoft FrontPage Express 2.0">
<-- <title>Bienvenue a l'ISTIA - Universite d'Angers</title>
<-- </head>
....
<-- face="Verdana"> - Dernire mise jour le <b>10 janvier 2002</b></font></p>
<-- </body>
<-- </html>
<--
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]

```

Un client Web envoie ses commandes au serveur selon le schéma suivant :

```

commande1
commande2
...
commanden
[ligne vide]

```

Ce n'est qu'après avoir reçu la ligne vide que le serveur Web répond. Dans l'exemple nous n'avons utilisé qu'une commande :

```
GET /index.html HTTP/1.0
```

qui demande au serveur l'URL **/index.html** et indique qu'il travaille avec le protocole HTTP version 1.0. La version la plus récente de ce protocole est 1.1. L'exemple montre que le serveur a répondu en renvoyant le contenu du fichier *index.html* puis qu'il a fermé la connexion puisqu'on voit le thread de lecture des réponses se terminer. Avant d'envoyer le contenu du fichier *index.html*, le serveur web a envoyé une série d'entêtes terminée par une ligne vide :

```

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>

```

La ligne *<html>* est la première ligne du fichier */index.html*. Ce qui précède s'appelle des entêtes HTTP (HyperText Transfer Protocol). Nous n'allons pas détailler ici ces entêtes mais on se rappellera que notre client générique y donne accès, ce qui peut être utile pour les comprendre. La première ligne par exemple :

```
<-- HTTP/1.1 200 OK
```

indique que le serveur Web contacté comprend le protocole HTTP/1.1 et qu'il a bien trouvé le fichier demandé (200 OK), 200 étant un code de réponse HTTP. Les lignes

```

<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html

```

disent au client qu'il va recevoir 11251 octets représentant du texte HTML (HyperText Markup Language) et qu'à la fin de l'envoi, la connexion sera fermée. On a donc là un client tcp très pratique. En fait, ce client existe déjà sur les machines où il s'appelle *telnet* mais il était intéressant de l'écrire nous-mêmes. Le programme du client tcp générique est le suivant :

```

' espaces de noms
Imports System
Imports System.Net.Sockets
Imports System.IO
Imports System.Threading
Imports Microsoft.VisualBasic

' la classe
Public Class clientTcpGénérique

```

```

' reçoit en paramètre les caractéristiques d'un service sous la forme
' serveur port
' se connecte au service
' crée un thread pour lire des commandes tapées au clavier
' celles-ci seront envoyées au serveur
' crée un thread pour lire les réponses du serveur
' celles-ci seront affichées à l'écran
' le tout se termine avec la commande fin tapée au clavier
Public Shared Sub Main(ByVal args() As String)

    ' syntaxe
    Const syntaxe As String = "pg serveur port"

    ' nombre d'arguments
    If args.Length <> 2 Then
        erreur(syntaxe, 1)
    End If

    ' on note le nom du serveur
    Dim serveur As String = args(0)

    ' le port doit être entier >0
    Dim port As Integer = 0
    Dim erreurPort As Boolean = False
    Dim E As Exception = Nothing
    Try
        port = Integer.Parse(args(1))
    Catch ex As Exception
        E = ex
        erreurPort = True
    End Try
    erreurPort = erreurPort Or port <= 0
    If erreurPort Then
        erreur(syntaxe + ControlChars.Lf + "Port incorrect (" + E.ToString + ")", 2)
    End If
    Dim client As TcpClient = Nothing
    ' il peut y avoir des problèmes
    Try
        ' on se connecte au service
        client = New TcpClient(serveur, port)
    Catch ex As Exception
        ' erreur
        Console.Error.WriteLine(("Impossible de se connecter au service (" & serveur & "," & port & "),
erreur : " & ex.Message))
        ' fin
        Return
    End Try

    ' on crée les threads de lecture/écriture
    Dim thReceive As New Thread(New ThreadStart(AddressOf New clientReceive(client).Run))
    Dim thSend As New Thread(New ThreadStart(AddressOf New clientSend(client).Run))

    ' on lance l'exécution des deux threads
    thSend.Start()
    thReceive.Start()

    ' fin du thread principal
    Return
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

Public Class clientSend
    ' classe chargée de lire des commandes tapées au clavier
    ' et de les envoyer à un serveur via un client tcp passé au constructeur
    Private client As TcpClient ' le client tcp

    ' constructeur
    Public Sub New(ByVal client As TcpClient)
        ' on note le client tcp
        Me.client = client
    End Sub

    ' méthode Run du thread

```

```

Public Sub Run()

    ' données locales
    Dim OUT As StreamWriter = Nothing ' flux d'écriture réseau
    Dim commande As String = Nothing ' commande lue au clavier
    ' gestion des erreurs
    Try
        ' création du flux d'écriture réseau
        OUT = New StreamWriter(client.GetStream())
        OUT.AutoFlush = True
        ' boucle saisie-envoi des commandes
        Console.Out.WriteLine("Commandes : ")
        While True
            ' lecture commande tapée au clavier
            commande = Console.In.ReadLine().Trim()
            ' fini ?
            If commande.ToLower() = "fin" Then
                Exit While
            End If
            ' envoi commande au serveur
            OUT.WriteLine(commande)
        End While
    Catch ex As Exception
        ' erreur
        Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
    End Try
    ' fin - on ferme les flux
    Try
        OUT.Close()
        client.Close()
    Catch
    End Try
    ' on signale la fin du thread
    Console.Out.WriteLine("[fin du thread d'envoi des commandes au serveur]")
End Sub
End Class

Public Class clientReceive
    ' classe chargée de lire les lignes de texte destinées à un
    ' client tcp passé au constructeur
    Private client As TcpClient ' le client tcp

    ' constructeur
    Public Sub New(ByVal client As TcpClient)
        ' on note le client tcp
        Me.client = client
    End Sub

    ' constructeur
    ' méthode Run du thread
    Public Sub Run()

        ' données locales
        Dim [IN] As StreamReader = Nothing ' flux lecture réseau
        Dim réponse As String = Nothing ' réponse serveur
        ' gestion des erreurs
        Try
            ' création du flux lecture réseau
            [IN] = New StreamReader(client.GetStream())
            ' boucle lecture lignes de texte du flux IN
            While True
                ' lecture flux réseau
                réponse = [IN].ReadLine()
                ' flux fermé ?
                If réponse Is Nothing Then
                    Exit While
                End If
                ' affichage
                Console.Out.WriteLine(("<-- " + réponse))
            End While
        Catch ex As Exception
            ' erreur
            Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
        End Try
        ' fin - on ferme les flux
        Try
            [IN].Close()
            client.Close()
        Catch
        End Try
    End Sub
End Class

```

```

' on signale la fin du thread
Console.Out.WriteLine("[fin du thread de lecture des réponses du serveur]")
End Sub
End Class

```

## 8.4.4 Un serveur Tcp générique

Maintenant nous nous intéressons à un serveur

- qui affiche à l'écran les commandes envoyées par ses clients
- leur envoie comme réponse les lignes de texte tapées au clavier par un utilisateur. C'est donc ce dernier qui fait office de serveur.

Le programme est lancé par : **srvgen portEcoule**, où *portEcoule* est le port sur lequel les clients doivent se connecter. Le service au client sera assuré par deux threads :

- un thread se consacrant exclusivement à la lecture des lignes de texte envoyées par le client
- un thread se consacrant exclusivement à la lecture des réponses tapées au clavier par l'utilisateur. Celui-ci signalera par la commande **fin** qu'il clôt la connexion avec le client.

Le serveur crée deux threads par client. S'il y a  $n$  clients, il y aura  $2n$  threads actifs en même temps. Le serveur lui ne s'arrête jamais sauf par un Ctrl-C tapé au clavier par l'utilisateur. Voyons quelques exemples.

Le serveur est lancé sur le port 100 et on utilise le client générique pour lui parler. La fenêtre du client est la suivante :

```

dos>cltgen localhost 100
Commandes :
commande 1 du client 1
<-- réponse 1 au client 1
commande 2 du client 1
<-- réponse 2 au client 1
fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du thread de lecture des réponses du serveur]
[fin du thread d'envoi des commandes au serveur]

```

Les lignes commençant par <-- sont celles envoyées du serveur au client, les autres celles du client vers le serveur. La fenêtre du serveur est la suivante :

```

dos>srvgen 100
Serveur générique lancé sur le port 100
Thread de lecture des réponses du serveur au client 1 lancé
1 : Thread de lecture des demandes du client 1 lancé
<-- commande 1 du client 1
réponse 1 au client 1
1 : <-- commande 2 du client 1
réponse 2 au client 1
1 : [fin du Thread de lecture des demandes du client 1]
fin
[fin du Thread de lecture des réponses du serveur au client 1]

```

Les lignes commençant par <-- sont celles envoyées du client au serveur. Les lignes  $N$  : sont les lignes envoyées du serveur au client n°  $N$ . Le serveur ci-dessus est encore actif alors que le client 1 est terminé. On lance un second client pour le même serveur :

```

dos>cltgen localhost 100
Commandes :
commande 3 du client 2
<-- réponse 3 au client 2
fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du thread de lecture des réponses du serveur]
[fin du thread d'envoi des commandes au serveur]

```

La fenêtre du serveur est alors celle-ci :

```

dos>srvgen 100
Serveur générique lancé sur le port 100
Thread de lecture des réponses du serveur au client 1 lancé
1 : Thread de lecture des demandes du client 1 lancé
<-- commande 1 du client 1
réponse 1 au client 1
1 : <-- commande 2 du client 1
réponse 2 au client 1
1 : [fin du Thread de lecture des demandes du client 1]
fin
[fin du Thread de lecture des réponses du serveur au client 1]

```

```

Thread de lecture des réponses du serveur au client 2 lancé
2 : Thread de lecture des demandes du client 2 lancé
<-- commande 3 du client 2
réponse 3 au client 2
2 : [fin du Thread de lecture des demandes du client 2]
fin
[fin du Thread de lecture des réponses du serveur au client 2]
^C

```

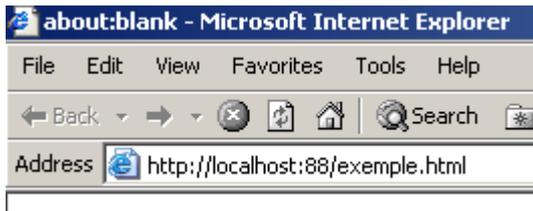
Simulons maintenant un serveur web en lançant notre serveur générique sur le port 88 :

```

dos>srvgen 88
Serveur générique lancé sur le port 88

```

Prenons maintenant un navigateur et demandons l'URL `http://localhost:88/exemple.html`. Le navigateur va alors se connecter sur le port 88 de la machine `localhost` puis demander la page `/exemple.html` :



Regardons maintenant la fenêtre de notre serveur :

```

dos>srvgen 88
Serveur générique lancé sur le port 88
Thread de lecture des réponses du serveur au client 2 lancé
2 : Thread de lecture des demandes du client 2 lancé
<-- GET /exemple.html HTTP/1.1
<-- Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, */*
<-- Accept-Language: fr
<-- Accept-Encoding: gzip, deflate
<-- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705; .NET CLR 1.0.2
914)
<-- Host: localhost:88
<-- Connection: Keep-Alive
<--

```

On découvre ainsi les entêtes HTTP envoyés par le navigateur. Cela nous permet de découvrir peu à peu le protocole HTTP. Lors d'un précédent exemple, nous avons créé un client Web qui n'envoyait que la seule commande GET. Cela avait été suffisant. On voit ici que le navigateur envoie d'autres informations au serveur. Elles ont pour but d'indiquer au serveur quel type de client il a en face de lui. On voit aussi que les entêtes HTTP se terminent par une ligne vide. Elaborons une réponse à notre client. L'utilisateur au clavier est ici le véritable serveur et il peut élaborer une réponse à la main. Rappelons-nous la réponse faite par un serveur Web dans un précédent exemple :

```

<-- HTTP/1.1 200 OK
<-- Date: Mon, 13 May 2002 07:30:58 GMT
<-- Server: Apache/1.3.12 (Unix) (Red Hat/Linux) PHP/3.0.15 mod_perl/1.21
<-- Last-Modified: Wed, 06 Feb 2002 09:00:58 GMT
<-- ETag: "23432-2bf3-3c60f0ca"
<-- Accept-Ranges: bytes
<-- Content-Length: 11251
<-- Connection: close
<-- Content-Type: text/html
<--
<-- <html>

```

Essayons de donner une réponse analogue :

```

...
<-- Host: localhost:88
<-- Connection: Keep-Alive
<--
2 : HTTP/1.1 200 OK
2 : Server: serveur tcp generique
2 : Connection: close
2 : Content-Type: text/html
2 :
2 : <html>
2 :   <head><title>Serveur generique</title></head>
2 :   <body>
2 :     <center>

```

```

2 :      <h2>Reponse du serveur generique</h2>
2 :      </center>
2 :      </body>
2 : </html>
2 : fin
L'erreur suivante s'est produite : Impossible de lire les données de la connexion de transport.
[fin du Thread de lecture des demandes du client 2]
[fin du Thread de lecture des réponses du serveur au client 2]

```

Les lignes commençant par 2 : sont envoyées du serveur au client n° 2. La commande *fin* clôt la connexion du serveur au client. Nous nous sommes limités dans notre réponse aux entêtes HTTP suivants :

```

HTTP/1.1 200 OK
2 : Server: serveur tcp generique
2 : Connection: close
2 : Content-Type: text/html
2 :

```

Nous ne donnons pas la taille du fichier que nous allons envoyer (*Content-Length*) mais nous contentons de dire que nous allons fermer la connexion (*Connection: close*) après envoi de celui-ci. Cela est suffisant pour le navigateur. En voyant la connexion fermée, il saura que la réponse du serveur est terminée et affichera la page HTML qui lui a été envoyée. Cette dernière est la suivante :

```

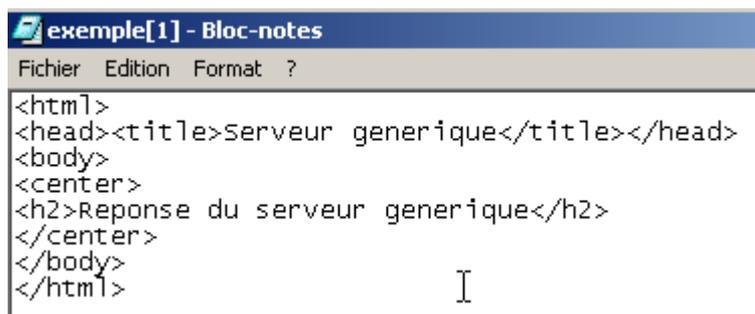
2 : <html>
2 :   <head><title>Serveur generique</title></head>
2 :   <body>
2 :     <center>
2 :       <h2>Reponse du serveur generique</h2>
2 :     </center>
2 :   </body>
2 : </html>

```

L'utilisateur ferme ensuite la connexion au client en tapant la commande *fin*. Le navigateur sait alors que la réponse du serveur est terminée et peut alors l'afficher :



Si ci-dessus, on fait *Affichage/Source* pour voir ce qu'a reçu le navigateur, on obtient :



c'est à dire exactement ce qu'on a envoyé depuis le serveur générique. Le code du serveur TCP générique est le suivant :

```

' espaces de noms
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.IO
Imports System.Threading
Imports Microsoft.VisualBasic

Public Class serveurTcpGénérique

' programme principal
Public Shared Sub Main(ByVal args() As String)

```

```

' reçoit le port d'écoute des demandes des clients
' crée un thread pour lire les demandes du client
' celles-ci seront affichées à l'écran
' crée un thread pour lire des commandes tapées au clavier
' celles-ci seront envoyées comme réponse au client
' le tout se termine avec la commande fin tapée au clavier

Const syntaxe As String = "Syntaxe : pg port"

' y-a-t-il un argument
If args.Length <> 1 Then
    erreur(syntaxe, 1)
End If
' cet argument doit être entier >0
Dim port As Integer = 0
Dim erreurPort As Boolean = False
Dim E As Exception = Nothing
Try
    port = Integer.Parse(args(0))
Catch ex As Exception
    E = ex
    erreurPort = True
End Try
erreurPort = erreurPort Or port <= 0
If erreurPort Then
    erreur(syntaxe + ControlChars.Lf + "Port incorrect (" + E.ToString + ")", 2)
End If
' on crée le servive d'écoute
Dim ecoute As TcpListener = Nothing
Dim nbClients As Integer = 0 ' nbre de clients traités
Try
    ' on crée le service
    ecoute = New TcpListener(IPAddress.Parse("127.0.0.1"), port)
    ' on le lance
    ecoute.Start()
    ' suivi
    Console.Out.WriteLine(("Serveur générique lancé sur le port " & port))

    ' boucle de service aux clients
    Dim client As TcpClient = Nothing
    While True ' boucle infinie - sera arrêtée par Ctrl-C
        ' attente d'un client
        client = ecoute.AcceptTcpClient()

        ' le service est assuré des threads séparés
        nbClients += 1
        ' thread de lecture des demandes clients
        Dim thReceive As New Thread(New ThreadStart(AddressOf New serveurReceive(client, nbClients).Run))
        ' thread de lecture des réponses tapées au clavier par l'utilisateur
        Dim thSend As New Thread(New ThreadStart(AddressOf New serveurSend(client, nbClients).Run))

        ' on lance l'exécution des deux threads
        thSend.Start()
        thReceive.Start()
    End While
    ' on retourne à l'écoute des demandes
Catch ex As Exception
    ' on signale l'erreur
    erreur("L'erreur suivante s'est produite : " + ex.Message, 3)
End Try
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

Public Class serveurSend
    ' classe chargée de lire des réponses tapées au clavier
    ' et de les envoyer à un client via un client tcp passé au constructeur
    Private client As TcpClient ' le client tcp
    Private numClient As Integer ' n° de client

    ' constructeur
    Public Sub New(ByVal client As TcpClient, ByVal numClient As Integer)

```

```

' on note le client tcp
Me.client = client
' et son n°
Me.numClient = numClient
End Sub

' méthode Run du thread
Public Sub Run()

' données locales
Dim OUT As StreamWriter = Nothing ' flux d'écriture réseau
Dim réponse As String = Nothing ' réponse lue au clavier
' suivi
Console.Out.WriteLine(("Thread de lecture des réponses du serveur au client " & numClient & "
lancé"))
' gestion des erreurs
Try
' création du flux d'écriture réseau
OUT = New StreamWriter(client.GetStream())
OUT.AutoFlush = True
' boucle saisie-envoi des commandes
While True
' identification client
Console.Out.Write((numClient & " : "))
' lecture réponse tapée au clavier
réponse = Console.In.ReadLine().Trim()
' fini ?
If réponse.ToLower() = "fin" Then
Exit While
End If
' envoi réponse au serveur
OUT.WriteLine(réponse)
End While
' réponse suivante
Catch ex As Exception
' erreur
Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
End Try
' fin - on ferme les flux
Try
OUT.Close()
client.Close()
Catch
End Try
' on signale la fin du thread
Console.Out.WriteLine(("[fin du Thread de lecture des réponses du serveur au client " & numClient &
"]"))
End Sub
End Class

Public Class serveurReceive
' classe chargée de lire les lignes de texte envoyées au serveur
' via un client tcp passé au constructeur
Private client As TcpClient ' le client tcp
Private numClient As Integer ' n° de client

' constructeur
Public Sub New(ByVal client As TcpClient, ByVal numClient As Integer)
' on note le client tcp
Me.client = client
' et son n°
Me.numClient = numClient
End Sub

' méthode Run du thread
Public Sub Run()
' données locales
Dim [IN] As StreamReader = Nothing ' flux lecture réseau
Dim réponse As String = Nothing ' réponse serveur
' suivi
Console.Out.WriteLine(("Thread de lecture des demandes du client " & numClient & " lancé"))
' gestion des erreurs
Try
' création du flux lecture réseau
[IN] = New StreamReader(client.GetStream())
' boucle lecture lignes de texte du flux IN
While True
' lecture flux réseau
réponse = [IN].ReadLine()
' flux fermé ?
If réponse Is Nothing Then

```

```

        Exit While
    End If
    ' affichage
    Console.Out.WriteLine("<-- " + réponse))
End While
Catch ex As Exception
    ' erreur
    Console.Error.WriteLine("L'erreur suivante s'est produite : " + ex.Message)
End Try
' fin - on ferme les flux
Try
    [IN].Close()
    client.Close()
Catch
End Try
' on signale la fin du thread
Console.Out.WriteLine("[fin du Thread de lecture des demandes du client " & numClient & "]")
End Sub
End Class

```

## 8.4.5 Un client Web

Nous avons vu dans l'exemple précédent, certains des entêtes HTTP qu'envoyait un navigateur :

```

<-- GET /exemple.html HTTP/1.1
<-- Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, */*
<-- Accept-Language: fr
<-- Accept-Encoding: gzip, deflate
<-- User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.0.3705; .NET CLR 1.0.2
914)
<-- Host: localhost:88
<-- Connection: Keep-Alive
<--

```

Nous allons écrire un client Web auquel on passerait en paramètre une URL et qui afficherait à l'écran le texte envoyé par le serveur. Nous supposons que celui-ci supporte le protocole HTTP 1.1. Des entêtes précédents, nous n'utiliserons que les suivants :

```

<-- GET /exemple.html HTTP/1.1
<-- Host: localhost:88
<-- Connection: close

```

- le premier entête indique quelle page nous désirons
- le second quel serveur nous interrogeons
- le troisième que nous souhaitons que le serveur ferme la connexion après nous avoir répondu.

Si ci-dessus, nous remplaçons GET par HEAD, le serveur ne nous enverra que les entêtes HTTP et pas la page HTML.

Notre client web sera appelé de la façon suivante : **clientweb URL cmd**, où **URL** est l'URL désirée et **cmd** l'un des deux mots clés GET ou HEAD pour indiquer si on souhaite seulement les entêtes (HEAD) ou également le contenu de la page (GET). Regardons un premier exemple. Nous lançons le serveur IIS puis le client web sur la même machine :

```

dos>clientweb http://localhost HEAD
HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.0
Date: Mon, 13 May 2002 09:23:37 GMT
Connection: close
Location: /IISamples/Default/welcome.htm
Content-Length: 189
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGQQGUUY=HMFNCCMDECBJJBPPBHAOAJNP; path=/
Cache-control: private

```

La réponse

```

HTTP/1.1 302 Object moved

```

signifie que la page demandée a changé de place (donc d'URL). La nouvelle URL est donnée par l'entête **Location:**

```

Location: /IISamples/Default/welcome.htm

```

Si nous utilisons GET au lieu de HEAD dans l'appel au client Web :

```

dos>clientweb http://localhost GET
HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.0
Date: Mon, 13 May 2002 09:33:36 GMT
Connection: close
Location: /IISamples/Default/welcome.htm
Content-Length: 189
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGQQGUUY=IMFNCCMDAKPNNMGGMFIHENFE; path=/
Cache-control: private

<head><title>L'objet a changé d'emplacement</title></head>
<body><h1>L'objet a changé d'emplacement</h1>< Cet objet peut être trouvé <a HREF="/IISamples/Default/we
lcome.htm">ici</a>.</body>

```

Nous obtenons le même résultat qu'avec HEAD avec de plus le corps de la page HTML. Le programme est le suivant :

```

' espaces de noms
Imports System
Imports System.Net.Sockets
Imports System.IO

Public Class clientWeb1

    ' demande une URL
    ' affiche le contenu de celle-ci à l'écran
    Public Shared Sub Main(ByVal args() As String)
        ' syntaxe
        Const syntaxe As String = "pg URI GET/HEAD"

        ' nombre d'arguments
        If args.Length <> 2 Then
            erreur(syntaxe, 1)
        End If

        ' on note l'URI demandée
        Dim URIstring As String = args(0)
        Dim commande As String = args(1).ToUpper()

        ' vérification validité de l'URI
        Dim uri As Uri = Nothing
        Try
            uri = New Uri(URIstring)
        Catch ex As Exception
            ' URI incorrecte
            erreur("L'erreur suivante s'est produite : " + ex.Message, 2)
        End Try

        ' vérification de la commande
        If commande <> "GET" And commande <> "HEAD" Then
            ' commande incorrecte
            erreur("Le second paramètre doit être GET ou HEAD", 3)
        End If

        ' on peut travailler
        Dim client As TcpClient = Nothing ' le client
        Dim [IN] As StreamReader = Nothing ' le flux de lecture du client
        Dim OUT As StreamWriter = Nothing ' le flux d'écriture du client
        Dim réponse As String = Nothing ' réponse du serveur
        Try
            ' on se connecte au serveur
            client = New TcpClient(uri.Host, uri.Port)

            ' on crée les flux d'entrée-sortie du client TCP
            [IN] = New StreamReader(client.GetStream())
            OUT = New StreamWriter(client.GetStream())
            OUT.AutoFlush = True

            ' on demande l'URL - envoi des entêtes HTTP
            OUT.WriteLine((commande + " " + uri.PathAndQuery + " HTTP/1.1"))
            OUT.WriteLine(("Host: " + uri.Host + ":" & uri.Port))
            OUT.WriteLine("Connection: close")
            OUT.WriteLine()

            ' on lit la réponse
            réponse = [IN].ReadLine()
            While Not (réponse Is Nothing)
                ' on traite la réponse
                Console.Out.WriteLine(réponse)
                ' on lit la réponse
                réponse = [IN].ReadLine()
            End While
        Catch ex As Exception
            erreur(ex.Message, 4)
        End Try
    End Sub

    Sub erreur(msg As String, n As Integer)
        Console.WriteLine(msg)
    End Sub
End Class

```

```

End While
' c'est fini
client.Close()
Catch e As Exception
' on gère l'exception
erreur(e.Message, 4)
End Try
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
' affichage erreur
System.Console.Error.WriteLine(msg)
' arrêt avec erreur
Environment.Exit(exitCode)
End Sub
End Class

```

La seule nouveauté dans ce programme est l'utilisation de la classe **Uri**. Le programme reçoit une URL (*Uniform Resource Locator*) ou URI (*Uniform Resource Identifier*) de la forme `http://serveur:port/cheminPageHTML?param1=val1;param2=val2;...`. La classe **Uri** nous permet de décomposer la chaîne de l'URL en ses différents éléments. Un objet **Uri** est construit à partir de la chaîne **UriString** reçue en paramètre :

```

' vérification validité de l'URI
Dim uri As Uri = Nothing
Try
uri = New Uri(UriString)
Catch ex As Exception
' URI incorrecte
erreur("L'erreur suivante s'est produite : " + ex.Message, 2)
End Try

```

Si la chaîne **Uri** reçue en paramètre n'est pas une **Uri** valide (absence du protocole, du serveur, ...), une exception est lancée. Cela nous permet de vérifier la validité du paramètre reçu. Une fois l'objet **Uri** construit, on a accès aux différents éléments de cette **Uri**. Ainsi si l'objet *uri* du code précédent a été construit à partir de la chaîne `http://serveur:port/cheminPageHTML?param1=val1;param2=val2;...` on aura :

**uri.Host**=*serveur*, **uri.Port**=*port*, **uri.Path**=*cheminPageHTML*, **uri.Query**=*param1=val1;param2=val2;...*, **uri.pathAndQuery**=*cheminPageHTML?param1=val1;param2=val2;...*, **uri.Scheme**=*http*.

## 8.4.6 Client Web gérant les redirections

Le client Web précédent ne gère pas une éventuelle redirection de l'URL qu'il a demandée. Le client suivant la gère.

1. il lit la première ligne des entêtes HTTP envoyés par le serveur pour vérifier si on y trouve la chaîne *302 Object moved* qui signale une redirection
2. il lit les entêtes suivants. S'il y a redirection, il recherche la ligne *Location: url* qui donne la nouvelle URL de la page demandée et note cette URL.
3. il affiche le reste de la réponse du serveur. S'il y a redirection, les étapes 1 à 3 sont répétées avec la nouvelle URL. Le programme n'accepte pas plus d'une redirection. Cette limite fait l'objet d'une constante qui peut être modifiée.

Voici un exemple :

```

dos>clientweb2 http://localhost GET
HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.0
Date: Mon, 13 May 2002 11:38:55 GMT
Connection: close
Location: /IISamples/Default/welcome.htm
Content-Length: 189
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGQQGUUY=PDGNCCMDNCAOFDMPHCJNPBAI; path=/
Cache-control: private

<head><title>L'objet a chang d'emplacement</title></head>
<body><h1>L'objet a chang d'emplacement</h1>Cet objet peut tre trouv <a HREF="/IISamples/Default/wel
lcome.htm">ici</a>.</body>

<--Redirection vers l'URL http://localhost:80/IISamples/Default/welcome.htm-->

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Connection: close
Date: Mon, 13 May 2002 11:38:55 GMT

```

```
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Mon, 16 Feb 1998 21:16:22 GMT
ETag: "0174e21203bbd1:978"
Content-Length: 4781
```

```
<html>

<head>
<title>Bienvenue dans le Serveur Web personnel</title>
</head>
....
</body>
</html>
```

Le programme est le suivant :

```
' espaces de noms
Imports System
Imports System.Net.Sockets
Imports System.IO
Imports System.Text.RegularExpressions
Imports Microsoft.VisualBasic

' classe client web
Public Class clientWeb

' demande une URL et affiche le contenu de celle-ci à l'écran
Public Shared Sub Main(ByVal args() As String)
' syntaxe
Const syntaxe As String = "pg URI GET/HEAD"

' nombre d'arguments
If args.Length <> 2 Then
erreur(syntaxe, 1)
End If
' on note l'URI demandée
Dim URIstring As String = args(0)
Dim commande As String = args(1).ToUpper()

' vérification validité de l'URI
Dim uri As Uri = Nothing
Try
uri = New Uri(URIstring)
Catch ex As Exception
' URI incorrecte
erreur("L'erreur suivante s'est produite : " + ex.Message, 2)
End Try 'catch
' vérification de la commande
If commande <> "GET" And commande <> "HEAD" Then
' commande incorrecte
erreur("Le second paramètre doit être GET ou HEAD", 3)
End If

' on peut travailler
Dim client As TcpClient = Nothing ' le client
Dim [IN] As StreamReader = Nothing ' le flux de lecture du client
Dim OUT As StreamWriter = Nothing ' le flux d'écriture du client
Dim réponse As String = Nothing ' réponse du serveur
Const nbRedirsMax As Integer = 1 ' pas plus d'une redirection acceptée
Dim nbRedirs As Integer = 0 ' nombre de redirections en cours
Dim premièreLigne As String ' lère ligne de la réponse
Dim redir As Boolean = False ' indique s'il y a redirection ou non
Dim locationString As String = "" ' la chaîne URI d'une éventuelle redirection
' expression régulière pour trouver une URL de redirection
Dim location As New Regex("^Location: (.+?)$")

' gestion des erreurs
Try
' on peut avoir plusieurs URL à demander s'il y a des redirections
While nbRedirs <= nbRedirsMax
' on se connecte au serveur
client = New TcpClient(uri.Host, uri.Port)

' on crée les flux d'entrée-sortie du client TCP
[IN] = New StreamReader(client.GetStream())
OUT = New StreamWriter(client.GetStream())
OUT.AutoFlush = True

' on envoie les entêtes HTTP pour demander l'URL
OUT.WriteLine((commande + " " + uri.PathAndQuery + " HTTP/1.1"))
```

```

OUT.WriteLine("Host: " + uri.Host + ":" & uri.Port))
OUT.WriteLine("Connection: close")
OUT.WriteLine()

' on lit la première ligne de la réponse
premièreLigne = [IN].ReadLine()
' écho écran
Console.Out.WriteLine (premièreLigne)

' redirection ?
If Regex.IsMatch(premièreLigne, "302 Object moved$") Then
    ' il y a une redirection
    redir = True
    nbRedirs += 1
End If

' entêtes HTTP suivants jusqu'à trouver la ligne vide signalant la fin des entêtes
Dim locationFound As Boolean = False
réponse = [IN].ReadLine()
While réponse <> ""
    ' on affiche la réponse
    Console.Out.WriteLine (réponse)
    ' s'il y a redirection, on recherche l'entête Location
    If redir And Not locationFound Then
        ' on compare la ligne à l'expression relationnelle location
        Dim résultat As Match = location.Match(réponse)
        If résultat.Success Then
            ' si on a trouvé on note l'URL de redirection
            locationString = résultat.Groups(1).Value
            ' on note qu'on a trouvé
            locationFound = True
        End If
    End If
    ' ligne suivante
    réponse = [IN].ReadLine()
End While

' lignes suivantes de la réponse
Console.Out.WriteLine (réponse)
réponse = [IN].ReadLine()
While Not (réponse Is Nothing)
    ' on affiche la réponse
    Console.Out.WriteLine (réponse)
    ' ligne suivante
    réponse = [IN].ReadLine()
End While

' on ferme la connexion
client.Close()
' a-t-on fini ?
If Not locationFound Or nbRedirs > nbRedirsMax Then
    Exit While
End If

' il y a une redirection à opérer - on construit la nouvelle Uri
URIstring = uri.Scheme + "://" & uri.Host & ":" & uri.Port & locationString
uri = New Uri(URIstring)
' suivi
Console.Out.WriteLine((ControlChars.Lf + "<--Redirection vers l'URL " + URIstring + "-->" +
ControlChars.Lf))
End While
Catch e As Exception
    ' on gère l'exception
    erreur(e.Message, 4)
End Try
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine (msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

```

## 8.4.7 Serveur de calcul d'impôts

Nous reprenons l'exercice IMPOTS déjà traité sous diverses formes. Rappelons la dernière mouture. Une classe `impôt` a été créée. Ses attributs sont trois tableaux de nombres :

```
Public Class impôt
' les données nécessaires au calcul de l'impôt
' proviennent d'une source extérieure
Private limites(), coeffR(), coeffN() As Double
```

La classe a deux constructeurs :

- un constructeur à qui on passe les trois tableaux de données nécessaires au calcul de l'impôt

```
// constructeur 1
Public Sub New(ByVal LIMITES() As Decimal, ByVal COEFFR() As Decimal, ByVal COEFFN() As Decimal)
' initialise les trois tableaux limites, coeffR, coeffN à partir
' des paramètres passés au constructeur
```

- un constructeur à qui on passe le nom DSN d'une base de données ODBC

```
' constructeur 2
Public Sub New(ByVal DSNimpots As String, ByVal Timpots As String, ByVal collimites As String, ByVal
colCoeffR As String, ByVal colCoeffN As String)
' initialise les trois tableaux limites, coeffR, coeffN à partir
' du contenu de la table Timpots de la base ODBC DSNimpots
' collimites, colCoeffR, colCoeffN sont les trois colonnes de cette table
' peut lancer une exception
```

Un programme de test avait été écrit :

```
dos>vbc /r:impots.dll testimpots.vb
```

```
dos>test mysql-impots timpots limites coeffr coeffn
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 200000
impôt=22506 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 2 200000
impôt=33388 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 3 200000
impôt=16400 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 3 300000
impôt=50082 F
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :n 3 200000
impôt=22506 F
```

Ici le programme de test et l'objet `impôt` étaient sur la même machine. Nous nous proposons de mettre le programme de test et l'objet `impôt` sur des machines différentes. Nous aurons une application client-serveur où l'objet `impôt` distant sera le serveur. La nouvelle classe s'appelle `ServeurImpots` et est dérivée de la classe `impôt` :

```
Public Class ServeurImpots
Inherits impôt

' attributs
Private portEcoute As Integer ' le port d'écoute des demandes clients
Private actif As Boolean ' état du serveur

' constructeur
Public Sub New(ByVal portEcoute As Integer, ByVal DSNimpots As String, ByVal Timpots As String, ByVal
collimites As String, ByVal colCoeffR As String, ByVal colCoeffN As String)
MyBase.New(DSNimpots, Timpots, collimites, colCoeffR, colCoeffN)
' on note le port d'écoute
Me.portEcoute = portEcoute
' pour l'instant inactif
actif = False
' crée et lance un thread de lecture des commandes tapées au clavier
' le serveur sera géré à partir de ces commandes
Dim threadLecture As Thread = New Thread(New ThreadStart(AddressOf admin))
threadLecture.Start()
End Sub
```

Le seul paramètre nouveau dans le constructeur est le port d'écoute des demandes des clients. Les autres paramètres sont passés directement à la classe de base `impôt`. Le serveur d'impôts est contrôlé par des commandes tapées au clavier. Aussi crée-t-on un thread pour lire ces commandes. Il y en aura deux possibles : `start` pour lancer le service, `stop` pour l'arrêter définitivement. La méthode `admin` qui gère ces commandes est la suivante :

```
Public Sub admin()
Services WEB
```

```

' lit les commandes d'administration du serveur tapées au clavier
' dans une boucle sans fin
Dim commande As String = Nothing
While True
    ' invite
    Console.Out.WriteLine("Serveur d'impôts>")
    ' lecture commande
    commande = Console.In.ReadLine().Trim().ToLower()
    ' exécution commande
    If commande = "start" Then
        ' actif ?
        If actif Then
            ' erreur
            Console.Out.WriteLine("Le serveur est déjà actif")
        Else
            ' on lance le service d'écoute
            Dim threadEcoute As Thread = New Thread(New ThreadStart(AddressOf ecoute))
            threadEcoute.Start()
        End If
    Else
        If commande = "stop" Then
            ' fin de tous les threads d'exécution
            Environment.Exit(0)
        Else
            ' erreur
            Console.Out.WriteLine("Commande incorrecte. Utilisez (start,stop)")
        End If
    End If
End While
End Sub

```

Si la commande tapée au clavier est *start*, un thread d'écoute des demandes clients est lancé. Si la commande tapée est *stop*, tous les threads sont arrêtés. Le thread d'écoute exécute la méthode *ecoute* :

```

Public Sub ecoute()
    ' thread d'écoute des demandes des clients
    ' on crée le service d'écoute
    Dim ecoute As TcpListener = Nothing
    Try
        ' on crée le service
        ecoute = New TcpListener(IPAddress.Parse("127.0.0.1"), portEcoute)
        ' on le lance
        ecoute.Start()
        ' suivi
        Console.Out.WriteLine(("Serveur d'écho lancé sur le port " & portEcoute))

        ' boucle de service
        Dim liaisonClient As TcpClient = Nothing
        While True ' boucle infinie
            ' attente d'un client
            liaisonClient = ecoute.AcceptTcpClient()
            ' le service est assuré par une autre tâche
            Dim threadClient As Thread = New Thread(New ThreadStart(AddressOf New
traiteClientImpots(liaisonClient, Me).Run))
            threadClient.Start()
        End While
        ' on retourne à l'écoute des demandes
    Catch ex As Exception
        ' on signale l'erreur
        erreur("L'erreur suivante s'est produite : " + ex.Message, 3)
    End Try
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub

```

On retrouve un serveur tcp classique écoutant sur le port *portEcoute*. Les demandes des clients sont traitées par la méthode *Run* d'un objet auquel on passe deux paramètres :

1. l'objet *TcpClient* qui va permettre d'atteindre le client
2. l'objet *impôt this* qui va donner accès à la méthode *this.calculer* de calcul de l'impôt.

```

' -----
' assure le service à un client du serveur d'impôts
Public Class traiteClientImpots

```

```

Private liaisonClient As TcpClient ' liaison avec le client
Private [IN] As StreamReader ' flux d'entrée
Private OUT As StreamWriter ' flux de sortie
Private objImpôt As impôt ' objet Impôt

' constructeur
Public Sub New(ByVal liaisonClient As TcpClient, ByVal objImpôt As impôt)
    Me.liaisonClient = liaisonClient
    Me.objImpôt = objImpôt
End Sub

```

La méthode *Run* traite les demandes des clients. Celles-ci peuvent avoir deux formes :

1. calcul marié(o/n) nbEnfants salaireAnnuel
2. fincalculs

La forme 1 permet le calcul d'un impôt, la forme 2 clôt la liaison client-serveur.

```

' méthode Run
Public Sub Run()
    ' rend le service au client
    Try
        ' flux d'entrée
        [IN] = New StreamReader(liaisonClient.GetStream())
        ' flux de sortie
        OUT = New StreamWriter(liaisonClient.GetStream())
        OUT.AutoFlush = True
        ' envoi d'un msg de bienvenue au client
        OUT.WriteLine("Bienvenue sur le serveur d'impôts")

        ' boucle lecture demande/écriture réponse
        Dim demande As String = Nothing
        Dim champs As String() = Nothing ' les éléments de la demande
        Dim commande As String = Nothing ' la commande du client : calcul ou fincalculs
        demande = [IN].ReadLine()
        While Not (demande Is Nothing)
            ' on décompose la demande en champs
            champs = Regex.Split(demande.Trim().ToLower(), "\s+")
            ' deux demandes acceptées : calcul et fincalculs
            commande = champs(0)
            Dim erreur As Boolean = False
            If commande <> "calcul" And commande <> "fincalculs" Then
                ' erreur client
                OUT.WriteLine("Commande incorrecte. Utilisez (calcul,fincalculs).")
            End If
            If commande = "calcul" Then
                calculerImpôt(champs)
            End If
            If commande = "fincalculs" Then
                ' msg d'au-revoir au client
                OUT.WriteLine("Au revoir...")
                ' libération des ressources
                Try
                    OUT.Close()
                    [IN].Close()
                    liaisonClient.Close()
                Catch
                End Try
            End Try
            ' fin
            Return
        End If
        ' nouvelle demande
        demande = [IN].ReadLine()
    End While
Catch e As Exception
    erreur("L'erreur suivante s'est produite (" + e.ToString + ")", 2)
End Try
End Sub

```

Le calcul de l'impôt est effectué par la méthode *calculerImpôt* qui reçoit en paramètre le tableau des champs de la demande faite par le client. La validité de la demande est vérifiée et éventuellement l'impôt calculé et renvoyé au client.

```

' calcul d'impôts
Public Sub calculerImpôt(ByVal champs() As String)
    ' traite la demande : calcul marié nbEnfants salaireAnnuel
    ' décomposée en champs dans le tableau champs
    Dim marié As String = Nothing
    Dim nbEnfants As Integer = 0
    Dim salaireAnnuel As Integer = 0

```

```

' validité des arguments
Try
' il faut au moins 4 champs
If champs.Length <> 4 Then
    Throw New Exception
End If
' marié
marié = champs(1)
If marié <> "o" And marié <> "n" Then
    Throw New Exception
End If
' enfants
nbEnfants = Integer.Parse(champs(2))
' salaire
salaireAnnuel = Integer.Parse(champs(3))
Catch
    OUT.WriteLine(" syntaxe : calcul marié(O/N) nbEnfants salaireAnnuel")
' fini
Exit Sub
End Try
' on peut calculer l'impôt
Dim impot As Long = objImpôt.calculer(marié = "o", nbEnfants, salaireAnnuel)
' on envoie la réponse au client
OUT.WriteLine(impot.ToString)
End Sub

' affichage des erreurs
Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
' affichage erreur
System.Console.Error.WriteLine(msg)
' arrêt avec erreur
Environment.Exit(exitCode)
End Sub

```

Cette classe est compilée par

```
dos>vbc /r:impots.dll /r:system.dll /t:library srvimpots.vb
```

où *impots.dll* contient le code de la classe *impôt*. Un programme de test pourrait être le suivant :

```

' espaces de noms
Imports System
Imports System.IO
Imports Microsoft.VisualBasic

Public Class testServeurImpots
    Public Shared syntaxe As String = "Syntaxe : pg port dsnImpots Timpots colLimites colCoeffR colCoeffN"

' programme principal
Public Shared Sub Main(ByVal args() As String)

' il faut 6 arguments
If args.Length <> 6 Then
    erreur(syntaxe, 1)
End If
' le port doit être entier >0
Dim port As Integer = 0
Dim erreurPort As Boolean = False
Dim E As Exception = Nothing
Try
    port = Integer.Parse(args(0))
Catch ex As Exception
    E = ex
    erreurPort = True
End Try
erreurPort = erreurPort Or port <= 0
If erreurPort Then
    erreur(syntaxe + ControlChars.Lf + "Port incorrect (" + E.ToString + ")", 2)
End If
' on crée le serveur d'impôts
Try
    Dim srvimpots As ServeurImpots = New ServeurImpots(port, args(1), args(2), args(3), args(4), args(5))
Catch ex As Exception
'erreur
    Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
End Try
End Sub

' affichage des erreurs

```

```

Public Shared Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Class

```

On passe au programme de test les données nécessaires à la construction d'un objet *ServeurImpots* et à partir de là il crée cet objet. Ce programme de test est compilé par :

```
dos>vbc /r:svimpots.dll /r:impots.dll testimpots.vb
```

Voici un premier test :

```

dos>testimpots 124 odbc-mysql-dbimpots impots limites coeffr coeffn
Serveur d'impôts>Serveur d'impôts>start
Serveur d'impôts>Serveur d'écho lancé sur le port 124
stop

```

La ligne

```
dos>testimpots 124 odbc-mysql-dbimpots impots limites coeffr coeffn
```

crée un objet *ServeurImpots* qui n'écoute pas encore les demandes des clients. C'est la commande **start** tapée au clavier qui lance cette écoute. La commande **stop** arrête le serveur. Utilisons maintenant un client. Nous utiliserons le client générique créé précédemment. Le serveur est lancé :

```

dos>testimpots 124 odbc-mysql-dbimpots impots limites coeffr coeffn
Serveur d'impôts>Serveur d'impôts>start
Serveur d'impôts>Serveur d'écho lancé sur le port 124

```

Le client générique est lancé dans une autre fenêtre Dos :

```

dos> clttcpgenerique localhost 124Commandes :
<-- Bienvenue sur le serveur d'impôts

```

On voit que le client a bien récupéré le message de bienvenue du serveur. On envoie d'autres commandes :

```

x
<-- Commande incorrecte. Utilisez (calcul,fincalculs).
calcul
<-- syntaxe : calcul marié(O/N) nbEnfants salaireAnnuel
calcul o 2 200000
<-- 22506
calcul n 2 200000
<-- 33388
fincalculs
<-- Au revoir...
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]

```

On retourne dans la fenêtre du serveur pour l'arrêter :

```

dos>testimpots 124 odbc-mysql-dbimpots impots limites coeffr coeffn
Serveur d'impôts>Serveur d'impôts>start
Serveur d'impôts>Serveur d'écho lancé sur le port 124
stop

```

# 9. Services Web

## 9.1 Introduction

Nous avons présenté dans le chapitre précédent plusieurs applications client-serveur tcp-ip. Dans la mesure où les clients et le serveur échangent des lignes de texte, ils peuvent être écrits en n'importe quel langage. Le client doit simplement connaître le protocole de dialogue attendu par le serveur. Les services Web sont des applications serveur tcp-ip présentant les caractéristiques suivantes :

- Elles sont hébergées par des serveurs web et le protocole d'échanges client-serveur est donc HTTP (HyperText Transport Protocol), un protocole au-dessus de TCP-IP.
- Le service Web a un protocole de dialogue standard quelque soit le service assuré. Un service Web offre divers services S1, S2, ..., Sn. Chacun d'eux attend des paramètres fournis par le client et rend à celui-ci un résultat. Pour chaque service, le client a besoin de savoir :
  - le nom exact du service Si
  - la liste des paramètres qu'il faut lui fournir et leur type
  - le type de résultat retourné par le service

Une fois, ces éléments connus, le dialogue client-serveur suit le même format quelque soit le service web interrogé. L'écriture des clients est ainsi normalisée.

- Pour des raisons de sécurité vis à vis des attaques venant de l'internet, beaucoup d'organisations ont des réseaux privés et n'ouvrent sur Internet que certains ports de leurs serveurs : essentiellement le port 80 du service web. Tous les autres ports sont verrouillés. Aussi les applications client-serveur telles que présentées dans le chapitre précédent sont-elles construites au sein du réseau privé (intranet) et ne sont en général pas accessibles de l'extérieur. Loger un service au sein d'un serveur web le rend accessible à toute la communauté internet.
- Le service Web peut être modélisé comme un objet distant. Les services offerts deviennent alors des méthodes de cet objet. Un client peut avoir accès à cet objet distant comme s'il était local. Cela cache toute la partie communication réseau et permet de construire un client indépendant de cette couche. Si celle-ci vient à changer, le client n'a pas à être modifié. C'est là un énorme avantage et probablement le principal atout des services Web.
- Comme pour les applications client-serveur tcp-ip présentées dans le chapitre précédent, le client et le serveur peuvent être écrits dans un langage quelconque. Ils échangent des lignes de texte. Celles-ci comportent deux parties :
  - les entêtes nécessaires au protocole HTTP
  - le corps du message. Pour une réponse du serveur au client, celui-ci est au format XML (eXtensible Markup Language). Pour une demande du client au serveur, le corps du message peut avoir plusieurs formes dont XML. La demande XML du client peut avoir un format particulier appelé SOAP (Simple Object Access Protocol). Dans ce cas, la réponse du serveur suit aussi le format SOAP.

## 9.2 Les navigateurs et XML

Les services Web envoient du XML à leurs clients. Les navigateurs peuvent réagir différemment à la réception de ce flux XML. Internet Explorer a une feuille de style prédéfinie qui permet de l'afficher. Netscape Communicator n'a pas lui cette feuille de style et n'affiche pas le code XML reçu. Il faut visualiser le code source de la page reçue pour avoir accès au XML. Voici un exemple. pour le code XML suivant :

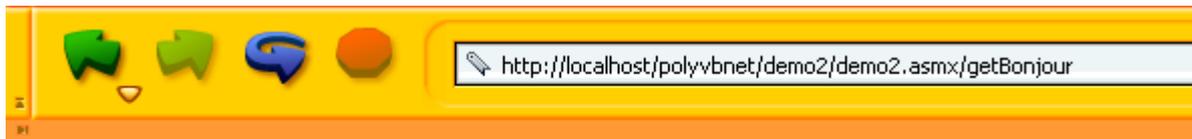
```
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="st.istia.univ-angers.fr">bonjour de nouveau !</string>
```

Internet Explorer affichera la page suivante :



```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="st.istia.univ-angers.fr">bonjour de nouveau !</string>
```

alors que Netscape Navigator affichera :



bonjour de nouveau !

Si on visualise le code source de la page reçue par Netscape, on obtient :

```
Source of: http://localhost/polyvbnet/demo2/demo2.asmx/getBonjour - Netscape
File Edit View Help
<?xml version="1.0" encoding="utf-8"?>
<string xmlns="st.istia.univ-angers.fr">bonjour de nouveau !</string>
```

Netscape a bien reçu la même chose que Internet Explorer mais il l'a affiché différemment. Dans la suite, nous utiliserons Internet Explorer pour les copies d'écran.

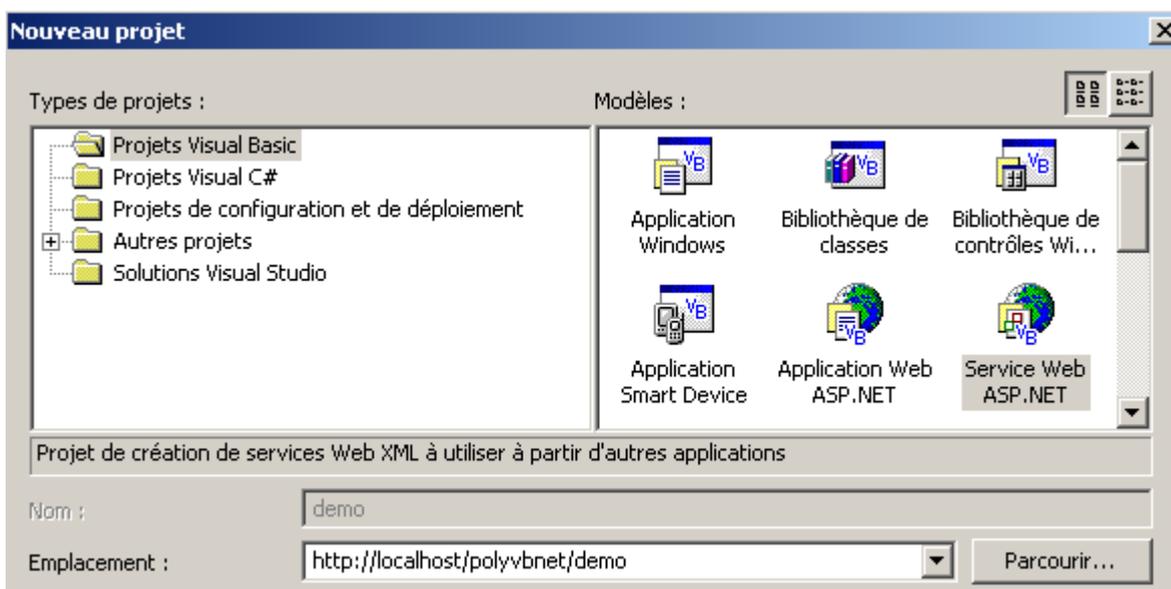
## 9.3 Un premier service Web

Nous allons découvrir les services web au travers d'un exemple simplissime décliné en trois versions.

### 9.3.1 Version 1

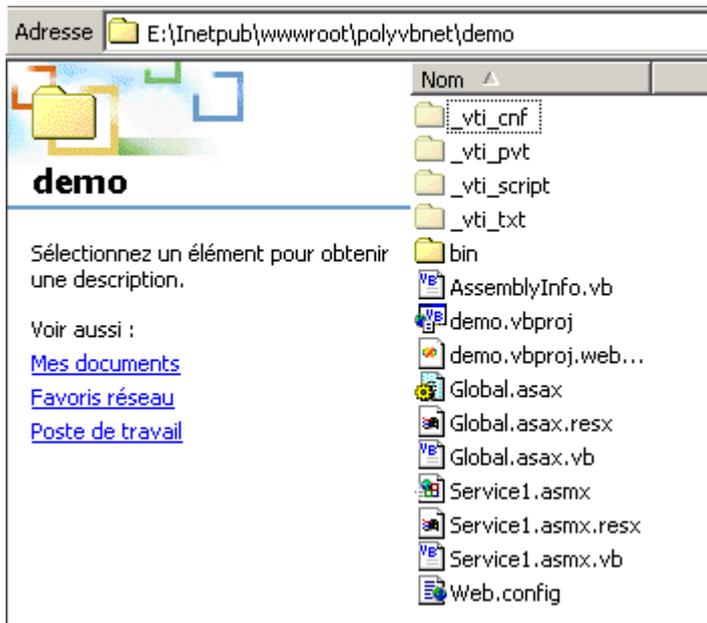
Pour cette première version nous allons utiliser VS.NET qui présente l'avantage de pouvoir générer un squelette de service web immédiatement opérationnel. Une fois comprise cette architecture, nous pourrons commencer à voler de nos propres ailes. Ce sera l'objet des versions suivantes.

Avec VS.NET, construisons un nouveau projet avec l'option [Fichier/Nouveau/Projet] :

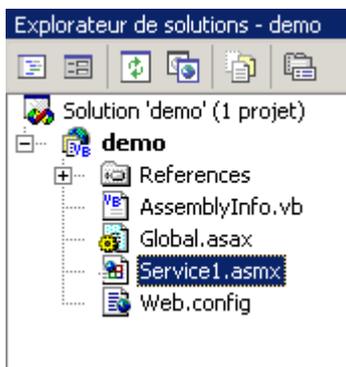


On notera les points suivants :

- le type du projet est Visual Basic (cadre de gauche)
- le modèle du projet est Service Web ASP.NET (cadre de droite)
- l'emplacement est libre. Ici, le service web sera hébergé par un serveur Web IIS local. Son URL sera donc `http://localhost/[chemin]` où [chemin] est à définir. Ici, nous choisissons le chemin `http://localhost/polyvbnet/demo`. VS.NET va alors créer un dossier pour ce projet. Où ? Le serveur IIS a une racine pour l'arborescence des documents web qu'il délivre. Appelons cette racine <IISroot>. Elle correspond à l'URL `http://localhost`. On en déduit que l'URL `http://localhost/polyvbnet/demo` sera associée au dossier <IISroot>/polyvbnet/demo. <IISroot> est normalement le dossier `\inetpub\wwwroot` sur le disque où a été installé IIS. Dans notre exemple c'est le disque E. Le dossier créé par VS.NET est donc le dossier `e:\inetpub\wwwroot\polyvbnet\demo` :



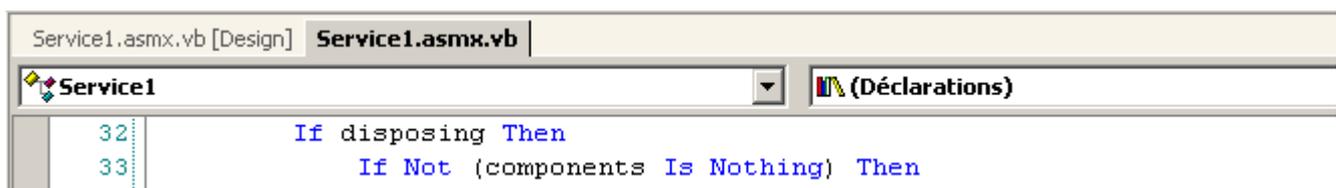
Comme toujours, il y a une surabondance de dossiers créés. Ils n'ont pas toujours un intérêt. Nous n'expliquerons que ceux dont nous avons besoin à un moment donné. VS.NET a créé un projet :



Nous retrouvons certains des fichiers présents dans le dossier physique du projet. Le plus intéressant pour nous est le fichier de suffixe **asmx**. C'est le suffixe des services web. Un service web est géré par VS.NET comme une application windows, c.a.d. une application qui a une interface graphique et du code pour la gérer. C'est pourquoi, nous avons une fenêtre de conception :



Un service web n'a normalement pas d'interface graphique. Il représente un objet qu'on peut appeler à distance. Il possède des méthodes et les applications appellent celles-ci. Nous le verrons donc comme un objet classique avec cette particularité qu'il a de pouvoir être instancié à distance via le réseau. Aussi, n'utiliserons-nous pas la fenêtre de conception présentée par VS.NET. Intéressons-nous plutôt au code du service en utilisant l'option Affichage/Code :



Plusieurs points sont à noter :

- le fichier s'appelle **Service1.asmx.vb** et non **Service1.asmx**. Nous reviendrons sur le contenu du fichier **Service1.asmx** un peu plus loin.

- on retrouve une fenêtre de code analogue à celle qu'on avait lorsqu'on construisait des applications windows avec VS.NET

Le code généré par VS.NET est le suivant :

```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace := "http://tempuri.org/demo/Service1")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    #Region " Code généré par le Concepteur des services Web "

        Public Sub New()
            MyBase.New()

            'Cet appel est requis par le Concepteur des services Web.
            InitializeComponent()

            'Ajoutez votre code d'initialisation après l'appel InitializeComponent()

        End Sub

        'Requis par le Concepteur des services Web
        Private components As System.ComponentModel.IContainer

        'REMARQUE : la procédure suivante est requise par le Concepteur des services Web
        'Elle peut être modifiée en utilisant le Concepteur des services Web.
        'Ne la modifiez pas en utilisant l'éditeur de code.
        <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
            components = New System.ComponentModel.Container()
        End Sub

        Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
            'CODEGEN : cette procédure est requise par le Concepteur des services Web
            'Ne la modifiez pas en utilisant l'éditeur de code.
            If disposing Then
                If Not (components Is Nothing) Then
                    components.Dispose()
                End If
            End If
            MyBase.Dispose(disposing)
        End Sub

    #End Region

    ' EXEMPLE DE SERVICE WEB
    ' L'exemple de service HelloWorld() retourne la chaîne Hello World.
    ' Pour générer, ne commentez pas les lignes suivantes, puis enregistrez et générez le projet.
    ' Pour tester ce service Web, assurez-vous que le fichier .asmx est la page de démarrage
    ' et appuyez sur F5.
    '
    '<WebMethod()> Public Function HelloWorld() As String
    ' HelloWorld = "Hello World"
    ' End Function

End Class
```

Tout d'abord, remarquons que nous avons là une classe, la classe **Service1** qui dérive de la classe **WebService** :

```
Public Class Service1
    Inherits System.Web.Services.WebService
```

Cela nous amène à importer l'espace de noms **System.Web.Services** :

```
Imports System.Web.Services
```

La déclaration de la classe est précédée d'un attribut de compilation :

```
<System.Web.Services.WebService(Namespace := "http://tempuri.org/demo/Service1")> _
Public Class Service1
    Inherits System.Web.Services.WebService
```

L'attribut `System.Web.Services.WebService()` indique que la classe qui suit est un service web. Cet attribut admet divers paramètres dont un appelé **NameSpace**. Il sert à placer le service web dans un espace de noms. En effet, on peut imaginer qu'il y ait plusieurs services web appelés **meteo** dans le monde. Il nous faut un moyen de les différencier. C'est l'espace de noms qui le permet. L'un

pourra s'appeler **[espace1].meteo** et un autre **[espace2].meteo**. On retrouve là, un concept analogue aux espaces de noms des classes. VS.NET a automatiquement généré du code qu'il a mis dans une région du source :

```
#Region " Code généré par le Concepteur des services Web "
```

Si on regarde ce code, on retrouve celui que le concepteur générerait lorsqu'on construisait des applications windows. C'est un code que l'on pourra purement et simplement supprimer si on n'a pas d'interface graphique, ce qui sera notre cas pour les services web.

La classe se termine par un exemple de ce que pourrait être un service web :

```
#End Region

' EXEMPLE DE SERVICE WEB
' L'exemple de service HelloWorld() retourne la chaîne Hello World.
' Pour générer, ne commentez pas les lignes suivantes, puis enregistrez et générez le projet.
' Pour tester ce service Web, assurez-vous que le fichier .asmx est la page de démarrage
' et appuyez sur F5.
'
'<WebMethod()> Public Function HelloWorld() As String
' HelloWorld = "Hello World"
' End Function
```

Fort de ce qui vient d'être dit, nous nettoyons le code pour qu'il devienne le suivant :

```
Imports System.Web.Services

<System.Web.Services.WebService(Namespace:="st.istia.univ-angers.fr")> _
Public Class Bonjour
    Inherits System.Web.Services.WebService

    <WebMethod()> Public Function Bonjour() As String
        Return "bonjour !"
    End Function
End Class
```

Nous y voyons un peu plus clair.

- un service web est une classe dérivant de la classe `WebService`
- la classe est qualifiée par l'attribut `<System.Web.Services.WebService(Namespace:="st.istia.univ-angers.fr")>`. On place donc notre service dans l'espace de noms **st.istia.univ-angers.fr**.
- les méthodes de la classe sont qualifiées par un attribut `<WebMethod()>` indiquant qu'on a affaire à une méthode qui peut être appelée à distance via le réseau

La classe assurant notre service web s'appelle donc **Bonjour** et a une seule méthode s'appelant elle-aussi **Bonjour** qui rend une chaîne de caractères. Nous sommes prêts pour un premier test.

- lançons le serveur web IIS si ce n'est fait
- utilisons l'option **Déboguer/Exécuter sans débogage**. VS.NET

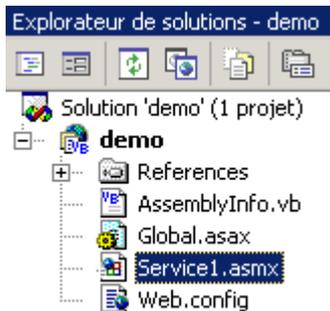
VS.NET va alors compiler l'ensemble de l'application, lancer un navigateur (souvent Internet Explorer s'il est présent), et afficher l'url **http://localhost/polyvbnet/demo/Service1.asmx** :



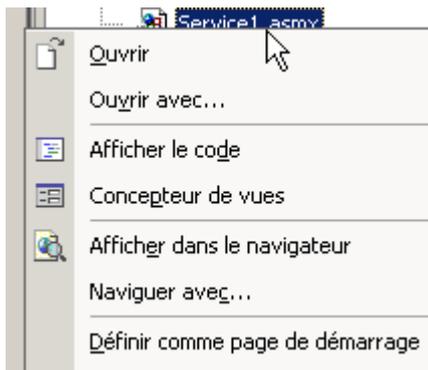
Les opérations suivantes sont prises en charge. Pour une définition formelle, prenez connaissance de [Description du service](#).

- [Bonjour](#)

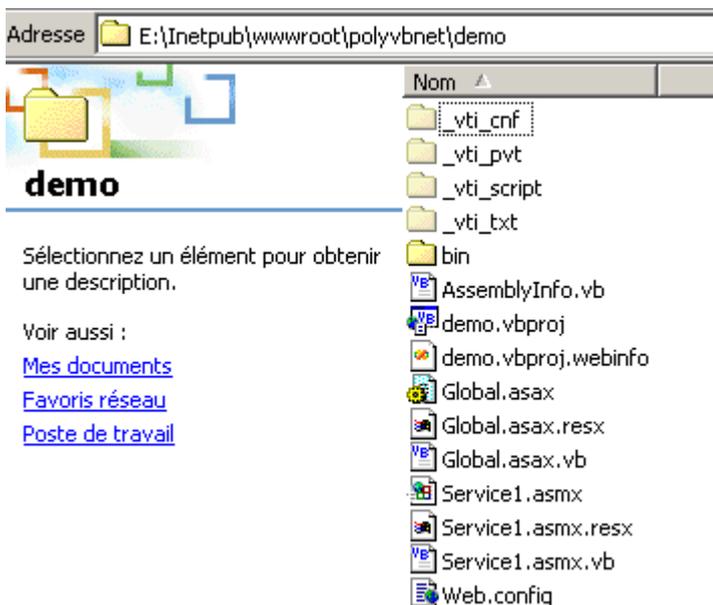
Pourquoi l'url **http://localhost/polyvbnet/demo/Service1.asmx** ? Parce que c'était le seul fichier .asmx du projet :



S'il y avait eu plusieurs fichiers .asmx, il nous aurait fallu préciser celui qui devait être exécuté en premier. Cela se fait en cliquant droit sur le fichier .asmx concerné et en prenant l'option [Définir comme page de démarrage].



On pourrait être intéressé par savoir ce que contient le fichier **service1.asmx**. En effet, avec VS.NET nous avons travaillé sur le fichier **service1.asmx.vb** et non sur le fichier **service1.asmx**. Ce fichier se trouve dans le dossier du projet :



Ouvrons avec un éditeur de texte (notepad ou autre). On obtient le contenu suivant :

```
<%@ WebService Language="vb" Codebehind="Service1.asmx.vb" Class="demo.Bonjour" %>
```

Le fichier contient une simple directive à l'intention du serveur IIS indiquant :

- qu'on a affaire à un service web (mot clé **WebService**)
- que le langage de la classe de ce service est Visual Basic (`Language="vb"`)
- que le source de cette classe sera trouvée dans le fichier `Service1.asmx.vb` (`Codebehind="Service1.asmx.vb"`)
- que la classe implémentant le service s'appelle `demo.Bonjour` (`Class="demo.Bonjour"`). On remarquera que VS.NET a placé la classe **Bonjour** dans l'espace de noms **demo** qui est aussi le nom du projet.

Revenons à la page obtenue à l'url <http://localhost/polyvbnet/demo/Service1.asmx> :



Les opérations suivantes sont prises en charge. Pour une définition formelle, prenez connaissance de [Description du service](#).

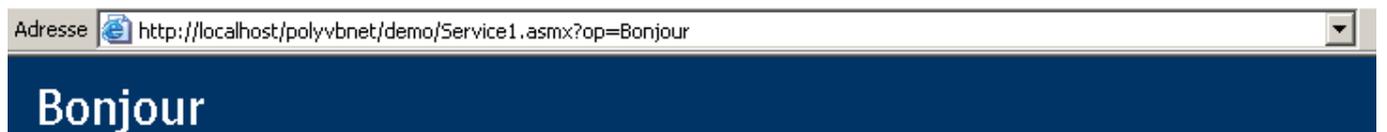
- [Bonjour](#)

Qui a écrit le code HTML de la page ci-dessus ? Pas nous, nous le savons. C'est IIS, qui présente les services web d'une façon standard. Cette page nous propose deux liens. Suivons le premier [Description du service] :



Ooops... c'est du XML plutôt abscons. Remarquons quand même l'URL

<http://localhost/polyvbnet/demo/Service1.asmx?WSDL>. Prenez un navigateur, et tapez directement cette url. Vous obtenez la même chose que précédemment. On se rappellera donc que l'url <http://serviceweb?WSDL> donne accès à la description XML du service web. Revenons à la page de départ et prenons le lien [Bonjour]. Rappelons-nous que Bonjour est une méthode du service web. Si nous avons eu plusieurs méthodes, elles auraient été toutes présentées ici. Nous obtenons la nouvelle page suivante :



Cliquez [ici](#) pour une liste complète des opérations.

## Bonjour

### Test

Pour tester l'opération en utilisant le protocole HTTP POST, cliquez sur le bouton 'Appeler'.



Nous avons volontairement tronqué la page obtenue pour ne pas alourdir notre démonstration. Remarquons de nouveau l'url obtenue :

<http://localhost/polyvbnet/demo/Service1.asmx?op=Bonjour>

Si nous tapons directement cette url dans un navigateur, nous obtiendrons la même chose que ci-dessus. On nous incite à utiliser le bouton [Appeler]. Faisons-le. Nous obtenons une nouvelle page :

Adresse  http://localhost/polyvbnet/demo/Service1.asmx/Bonjour

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="st.istia.univ-angers.fr">bonjour !</string>
```

C'est de nouveau du XML. On y retrouve deux informations qui étaient présentes dans notre service web :

- l'espace de noms `st.istia.univ-angers.fr` de notre service  
`<System.Web.Services.WebService (Namespace:="st.istia.univ-angers.fr")>`
- la valeur rendue par la méthode `Bonjour` :  
`Return "bonjour !"`

Qu'avons-nous appris ?

- la façon d'écrire un service web S
- la façon de l'appeler

Nous nous intéressons maintenant à l'écriture d'un service web sans l'aide de VS.NET.

## 9.3.2 Version 2

Dans l'exemple précédent, VS.NET a fait beaucoup de choses tout seul. Est-il possible de construire un service web sans cet outil ? La réponse est oui et nous le montrons maintenant. Avec un éditeur de texte, nous construisons le service web suivant :

```
Imports System.Web.Services
<System.Web.Services.WebService (Namespace:="st.istia.univ-angers.fr")> _
Public Class Bonjour2
  Inherits System.Web.Services.WebService
  <WebMethod()> Public Function getBonjour() As String
    Return "bonjour de nouveau !"
  End Function
End Class
```

La classe s'appelle **Bonjour2** et a une méthode qui s'appelle **getBonjour**. Elle a été placée dans le fichier **demo2.vb** lui-même placé dans l'arborescence du serveur IIS dans le dossier `E:\inetpub\wwwroot\polyvbnet\demo2`. C'est une classe VB.NET classique qu'on peut donc compiler :

```
dos>vbc /out:demo2 /t:library /r:system.dll /r:system.web.services.dll demo2.vb
dos>dir
02/03/2004 18:04          286 demo2.vb
02/03/2004 18:10           77 demo2.asmx
02/03/2004 18:12          3 072 demo2.dll
```

Nous mettons l'assemblage **demo2.dll** dans un dossier **bin** (ce nom est obligatoire) :

```
dos>dir bin
02/03/2004 18:12          3 072 demo2.dll
```

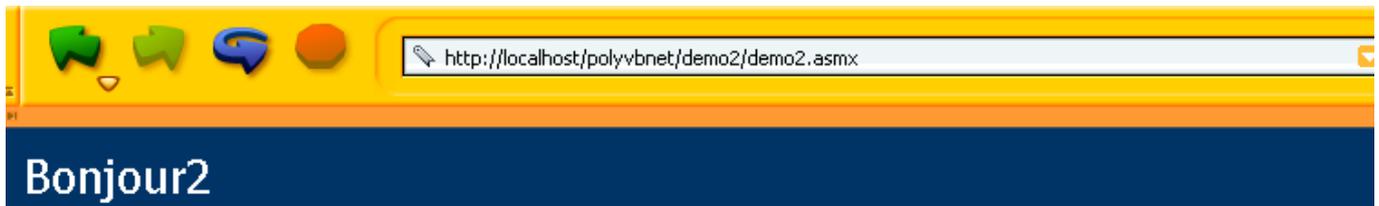
Nous créons maintenant le fichier **demo2.asmx**. C'est lui qui sera appelé par les clients web. Son contenu est le suivant :

```
<%@ WebService Language="vb" class="Bonjour2,demo2"%>
```

Nous avons déjà rencontré cette directive. Elle indique que :

- la classe du service web s'appelle **Bonjour2** et se trouve dans l'assemblage **demo2.dll**. IIS cherchera cet assemblage dans différents endroits et notamment dans le dossier **bin** du service web. C'est pourquoi, nous avons placé là l'assemblage **demo2.dll**.

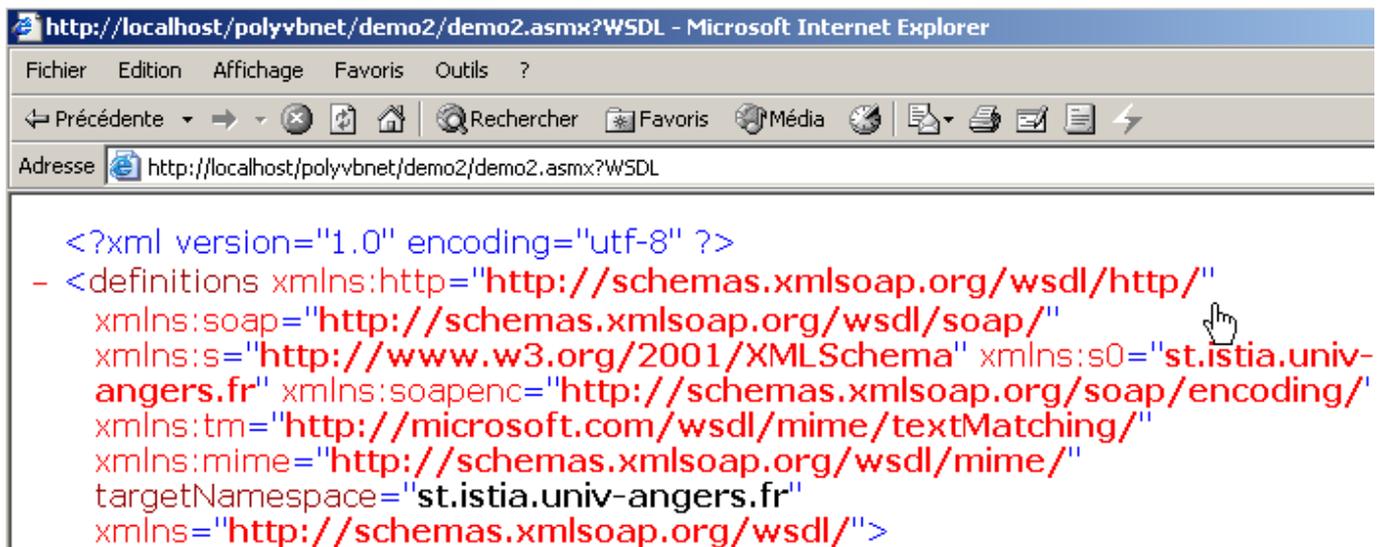
Maintenant nous pouvons faire divers tests. On s'assure que IIS est actif et on demande avec un navigateur l'url `http://localhost/polyvbnet/demo2/demo2.asmx` :



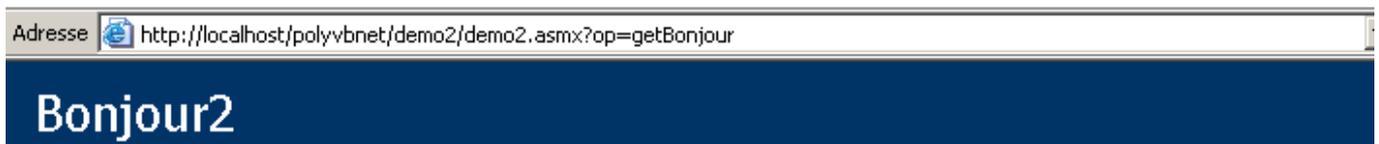
Les opérations suivantes sont prises en charge. Pour une définition formelle, prenez connaissance de [Description du service](#).

- ◆ [getBonjour](#)

Puis l'URL <http://localhost/polyvbnet/demo2/demo2.asmx?WSDL>



Puis l'URL <http://localhost/polyvbnet/demo2/demo2.asmx?op=getBonjour>, où `getBonjour` est le nom de l'unique méthode de notre service web :



Cliquez [ici](#) pour une liste complète des opérations.

## getBonjour

### Test

Pour tester l'opération en utilisant le protocole HTTP POST, cliquez sur le bouton 'Appeler'.



Nous utilisons le bouton [Appeler] ci-dessus :



Nous obtenons bien le résultat de l'appel à la méthode **getBonjour** du service web. Nous savons maintenant comment construire un service web sans vs.net. Nous ferons désormais abstraction de la façon dont est construit le service web pour ne nous intéresser qu'aux fichiers fondamentaux.

### 9.3.3 Version 3

Les deux versions précédentes du service web [Bonjour] utilisaient deux fichiers :

- un fichier .asmx, point d'entrée du service web
- un fichier .vb, code source du service web

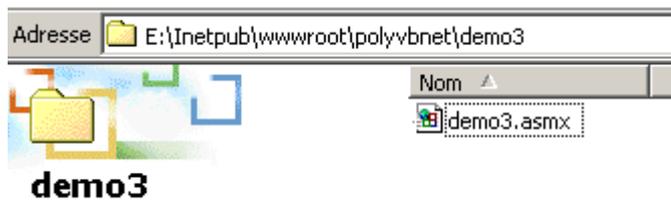
Nous montrons ici, qu'on peut se contenter du seul fichier .asmx. Le code du service **demo3.asmx** est le suivant :

```
<%@ WebService Language="vb" class="Bonjour3"%>
Imports System.Web.Services
<System.Web.Services.WebService(Namespace:="st.istia.univ-angers.fr")> _
Public Class Bonjour3
    Inherits System.Web.Services.WebService
    <WebMethod()> Public Function getBonjour() As String
        Return "bonjour en version3 !"
    End Function
End Class
```

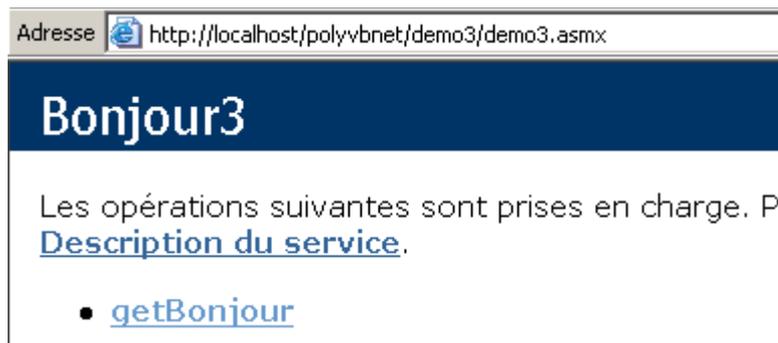
Nous constatons que le code source du service est maintenant directement dans le fichier source du fichier **demo3.asmx**. La directive

```
<%@ WebService Language="vb" class="Bonjour3"%>
```

ne référence plus une classe dans un assemblage externe, mais une classe se trouvant dans le même fichier source. Plaçons celui-ci dans le dossier <IISroot>\polyvbnet\demo3 :



Lançons IIS et demandons l'url **http://localhost/polyvbnet/demo3/demo3.asmx** :



Nous constatons une différence importante par rapport à la version précédente : nous n'avons pas eu à compiler le code VB du service. IIS a opéré cette compilation lui-même par l'intermédiaire du compilateur VB.NET installé sur la même machine. Puis il a délivré la page. S'il y a une erreur de compilation, celle-ci sera signalée par IIS :

Adresse  http://localhost/polyvbnet/demo3/demo3.aspx

**Message d'erreur du compilateur:** BC30451: Le nom 'Returnxx' n'est pas déclaré.

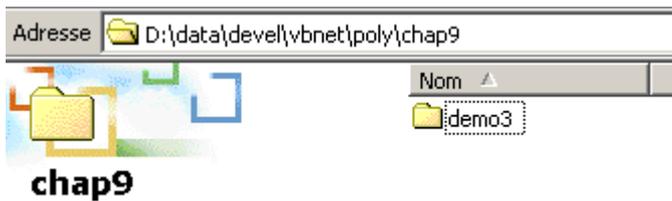
**Erreur source:**

```
Ligne 8 :  
Ligne 9 :     <WebMethod()> Public Function getBonjour() As String  
Ligne 10 :         Returnxx "bonjour en version3 !"  
Ligne 11 :     End Function  
Ligne 12 : End Class
```

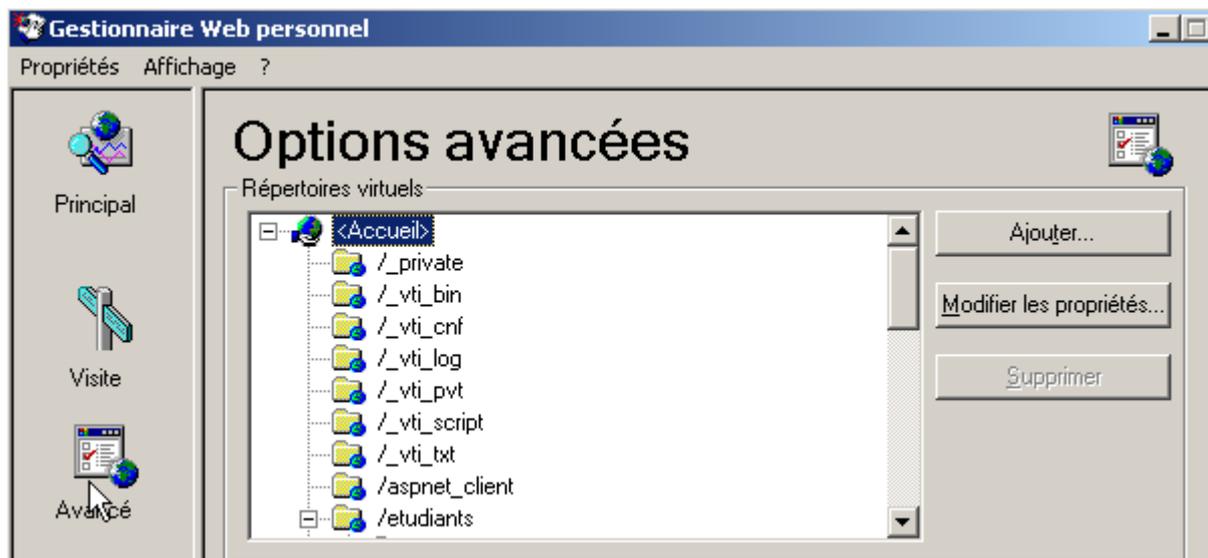
**Fichier source:** e:\inetpub\wwwroot\polyvbnet\demo3\demo3.aspx **Ligne:** 10

### 9.3.4 Version 4

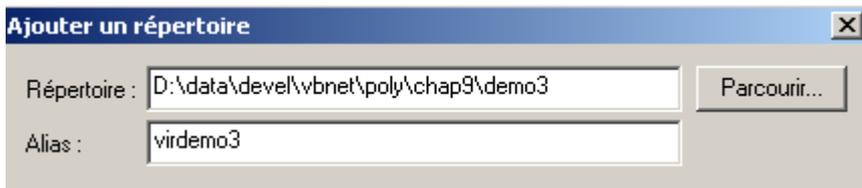
Nous nous intéressons ici à la configuration du serveur IIS. Nous avons toujours, jusqu'à maintenant, placé nos services web dans l'arborescence de racine <IISroot> du serveur IIS, ici [e:\inetpub\wwwroot]. Nous montrons ici que nous pouvons placer le service web n'importe où. Cela se fait à l'aide des dossiers virtuels de IIS. Plaçons notre service dans le dossier suivant :



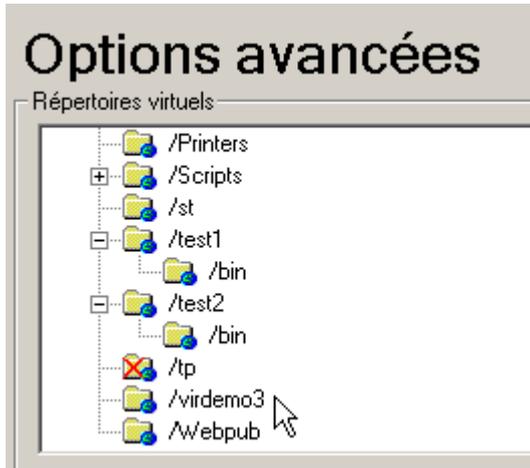
Le dossier [D:\data\devel\vbnet\poly\chap9\demo3] ne se trouve pas dans l'arborescence du serveur IIS. On doit l'indiquer à celui-ci en créant un dossier IIS virtuel. Lançons IIS et prenons l'option [Avancé] ci-dessous :



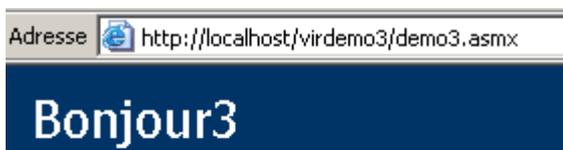
Nous avons une liste de répertoires virtuels qui nous est présentée. Nous ne nous attarderons pas sur celle-ci. On crée un nouveau répertoire virtuel avec le bouton [Ajouter] ci-dessus :



A l'aide du bouton [Parcourir], nous désignons le dossier physique contenant le service web, ici le dossier [D:\data\devel\vbnet\poly\chap9\demo3]. Nous donnons un nom logique (virtuel) à ce dossier : [virdemo3]. Cela signifie que les documents à l'intérieur du dossier physique [D:\data\devel\vbnet\poly\chap9\demo3] seront accessibles sur le réseau via l'url [http://<machine>/virdemo3]. La boîte de dialogue ci-dessus comporte d'autres paramètres qu'on laisse en l'état. Nous validons la boîte. Le nouveau dossier virtuel apparaît dans la liste des dossiers virtuels de IIS :



Maintenant, nous prenons un navigateur et nous demandons l'url [http://localhost/virdemo3/demo3.asmx]. Nous obtenons la même chose qu'auparavant :



Les opérations suivantes sont prises  
[Description du service.](#)

- [getBonjour](#)

### 9.3.5 Conclusion

Nous avons montré plusieurs façons de procéder pour créer un service web. Par la suite, nous utiliserons la méthode de la version 3 pour la création du service et la méthode 4 pour sa localisation. Nous n'aurons pas ainsi besoin de VS.NET. Néanmoins notons l'intérêt d'utiliser VS.NET pour l'aide qu'il apporte au débogage. Il existe des outils gratuits pour développer des application web, notamment le produit WebMatrix sponsorisé par Microsoft et qu'on trouvera à l'URL [http://www.asp.net/webmatrix]. C'est un outil excellent pour démarrer la programmation web sans investissement préalable.

## 9.4 Un service web d'opérations

Nous considérons un service Web qui offre cinq fonctions :

1. ajouter(a,b) qui rendra a+b
2. soustraire(a,b) qui rendra a-b
3. multiplier(a,b) qui rendra a\*b
4. diviser(a,b) qui rendra a/b
5. toutfaire(a,b) qui rendra le tableau [a+b,a-b,a\*b,a/b]

Le code VB.NET de ce service est le suivant :

```

<%@ WebService language="VB" class=operations %>

imports system.web.services

<WebService (Namespace:="st.istia.univ-angers.fr")> _
  Public Class operations
    Inherits WebService

    <WebMethod> _
    Function ajouter(a As Double, b As Double) As Double
      Return a + b
    End Function

    <WebMethod> _
    Function soustraire(a As Double, b As Double) As Double
      Return a - b
    End Function

    <WebMethod> _
    Function multiplier(a As Double, b As Double) As Double
      Return a * b
    End Function

    <WebMethod> _
    Function diviser(a As Double, b As Double) As Double
      Return a / b
    End Function

    <WebMethod> _
    Function toutfaire(a As Double, b As Double) As Double()
      Return New Double() {a + b, a - b, a * b, a / b}
    End Function
  End Class

```

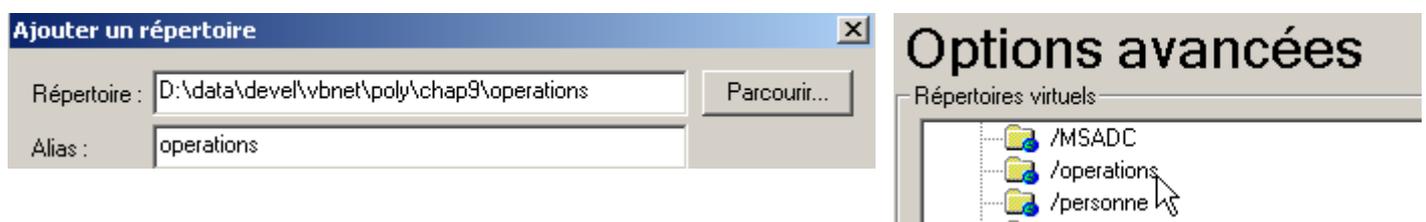
Nous reprenons ici certaines explications déjà données mais qui méritent d'être rappelées ou complétées. La classe *operations* ressemble à une classe VB.NET avec cependant quelques points à noter :

- les méthodes sont précédées d'un attribut **<WebMethod()>** qui indique au compilateur les méthodes qui doivent être "publiées" c.a.d. rendues disponibles au client. Une méthode non précédée de cet attribut serait invisible aux clients distants. Ce pourrait être une méthode interne utilisée par d'autres méthodes mais pas destinée à être publiée.
- la classe dérive de la classe *WebService* définie dans l'espace de noms *System.Web.Services*. Cet héritage n'est pas toujours obligatoire. Dans cet exemple notamment on pourrait s'en passer.
- la classe elle-même est précédée d'un attribut **<WebService(Namespace="st.istia.univ-angers.fr")>** destiné à donner un espace de noms au service web. Un vendeur de classes donne un espace de noms à ses classes afin de leur donner un nom unique et éviter ainsi des conflits avec des classes d'autres vendeurs qui pourraient porter le même nom. Pour les services Web, c'est pareil. Chaque service web doit pouvoir être identifié par un nom unique, ici par **st.istia.univ-angers.fr**.
- nous n'avons pas défini de constructeur. C'est donc implicitement le constructeur de la classe parent qui sera utilisé.

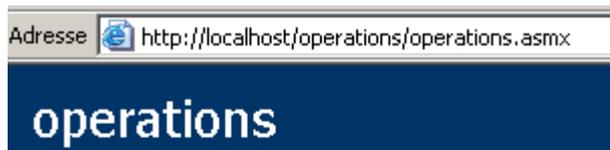
Le code source précédent n'est pas destiné directement au compilateur VB.NET mais au serveur Web IIS. Il doit porter le suffixe *.asmx* et sauvegardé dans l'arborescence du serveur Web. Ici nous le sauvegardons sous le nom **operations.asmx** dans le dossier **<IISroot>\polyvbnet\operations** :



Nous associons à ce dossier physique, le dossier virtuel IIS [operations] :



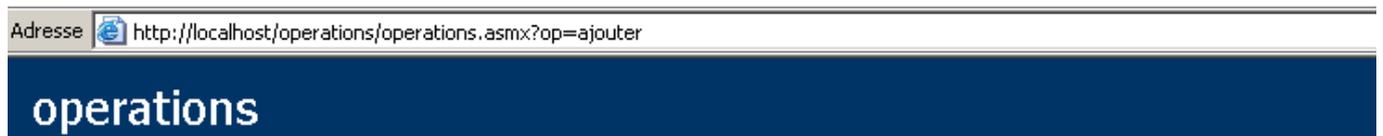
Accédons au service avec un navigateur. L'URI à demander est [http://localhost/operations/operations.asmx] :



Les opérations suivantes sont prises en

- [diviser](#)
- [multiplier](#)
- [ajouter](#)
- [soustraire](#)
- [toutfaire](#)

Nous obtenons un document Web avec un lien pour chacune des méthodes définies dans le service web *operations*. Suivons le lien *ajouter* :



Cliquez [ici](#) pour une liste complète des opérations.

## ajouter

### Test

Pour tester l'opération en utilisant le protocole HTTP POST, cliquez sur le bouton 'Appeler'.

Paramètre	Valeur
a:	<input type="text" value="15"/>
b:	<input type="text" value="30"/>

La page obtenue nous propose de tester la méthode *ajouter* en lui fournissant les deux arguments *a* et *b* dont elle a besoin. Rappelons la définition de la méthode *ajouter* :

```
<WebMethod>  
Function ajouter(a As Double, b As Double) As Double  
    Return a + b  
End Function
```

On notera que la page a repris les noms des arguments *a* et *b* utilisés dans la définition de la méthode. On utilise le bouton *Appeler* et on obtient la réponse suivante dans une fenêtre séparée du navigateur :

Adresse  http://localhost/operations/operations.asmx/ajouter

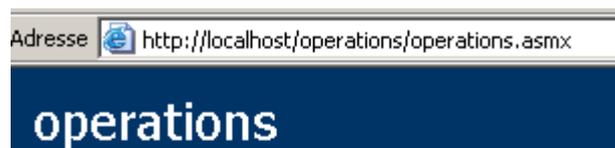
```
<?xml version="1.0" encoding="utf-8" ?>  
<double xmlns="st.istia.univ-angers.fr">45</double>
```

Si ci-dessus, on fait [Affichage/Source] on obtient le code suivant :



```
ajouter[1] - Bloc-notes  
Fichier Edition Format ?  
<?xml version="1.0" encoding="utf-8"?>  
<double xmlns="st.istia.univ-angers.fr">45</double>
```

Refaisons l'opération pour la méthode [toutfaire] :



Les opérations suivantes sont prises en

- [diviser](#)
- [multiplier](#)
- [ajouter](#)
- [soustraire](#)
- [toutfaire](#) 

Nous obtenons la page suivante :

# operations

Cliquez [ici](#) pour une liste complète des opérations.

## toutfaire

### Test

Pour tester l'opération en utilisant le protocole HTTP POST, cliquez sur le bouton

Paramètre	Valeur
a:	<input type="text" value="10"/>
b:	<input type="text" value="20"/>

Utilisons le bouton [Appeler] ci-dessus :

```
<?xml version="1.0" encoding="utf-8" ?>  
- <ArrayOfDouble xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns="st.istia.univ-angers.fr">  
  <double>30</double>  
  <double>-10</double>  
  <double>200</double>  
  <double>0.5</double>  
</ArrayOfDouble>
```

Dans tous les cas, la réponse du serveur a la forme :

```
<?xml version="1.0" encoding="utf-8"?>
```

[réponse au format XML]

- la réponse est au format XML
- la ligne 1 est standard et est toujours présente dans la réponse
- les lignes suivantes dépendent du type de résultat (**double**, **ArrayOfDouble**), du nombre de résultats, et de l'espace de noms du service web (**st.istia.univ-angers.fr** ici).

Il existe plusieurs méthodes interroger un service web et obtenir sa réponse . Revenons à l'URL du service :



et suivons le lien [ajouter]. Dans page présentée, sont exposées deux méthodes pour interroger la fonction [ajouter] du service web :

## HTTP POST

Le texte suivant est un exemple de demande et de réponse HTTP POST. Les **espaces réservés** affichés doivent être remplacés par des valeurs réelles.

```
POST /operations/operations.asmx/ajouter HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

a=string&b=string
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<double xmlns="st.istia.univ-angers.fr">double</double>
```



## SOAP

Le texte suivant est un exemple de demande et de réponse SOAP. Les **espaces réservés** affichés doivent être remplacés par des valeurs réelles.

```
POST /operations/operations.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "st.istia.univ-angers.fr/ajouter"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.o
  <soap:Body>
    <ajouter xmlns="st.istia.univ-angers.fr">
      <a>double</a>
      <b>double</b>
    </ajouter>
  </soap:Body>
</soap:Envelope>
```

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
  <soap:Body>
    <ajouterResponse xmlns="st.istia.univ-angers.fr">
      <ajouterResult>double</ajouterResult>
    </ajouterResponse>
  </soap:Body>
</soap:Envelope>

```

Ces deux méthodes d'accès aux fonctions d'un service web sont appelées respectivement : **HTTP-POST** et **SOAP**. Nous les examinons maintenant l'une après l'autre.

**Note** : dans les premières versions de VS.NET, il existait une 3<sup>ème</sup> méthode appelée **HTTP-GET**. Au jour d'écriture de ce document (mars 2004), cette méthode ne semble plus être disponible. Cela veut dire que le service web généré par VS.NET n'accepte pas de requêtes GET. Cela ne veut pas dire qu'on ne peut pas écrire de services web acceptant les requêtes GET, notamment avec d'autres outils que VS.NET ou simplement à la main.

## 9.5 Un client HTTP-POST

Nous suivons la méthode proposée par le service web :

### HTTP POST

Le texte suivant est un exemple de demande et de réponse HTTP POST. Les **espaces réservés** affichés doivent être remplacés par des valeurs réelles.

```

POST /operations/operations.asmx/ajouter HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: length

```

**a=string&b=string**

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<double xmlns="st.istia.univ-angers.fr">double</double>

```

Commentons ce qui est écrit. Tout d'abord le client web doit envoyer les entêtes HTTP suivants :

POST /operations/operations.asmx/ajouter HTTP/1.1

Le client web fait une requête POST à l'URL /operations/operations.asmx/ajouter selon le protocole HTTP version 1.1

HOST: localhost

On précise la machine cible de la requête. Ici localhost. Cet entête a été rendu obligatoire par la version 1.1 du protocole HTTP

Content-Type: application/x-www-form-urlencoded

On précise ici qu'après les entêtes HTTP on va envoyer des paramètres supplémentaires au format **urlencodé**. Ce format remplace certains caractères par leur code hexadécimal.

Content-length: 7

C'est la taille en caractères de la chaîne de paramètres qui sera envoyée après les entêtes HTTP.

Les entêtes HTTP sont suivis d'une ligne vide puis de la chaîne de paramètres du POST de [Content-Length] caractères sous la forme a=XX&b=YY où XX et YY sont les chaînes "urlencodées" des valeurs des paramètres a et b. Nous en savons assez pour reproduire ce qui ci-dessus avec notre client tcp générique déjà utilisé dans le chapitre sur la programmation tcp-ip :

- nous lançons IIS
- le service est disponible à l'url [http://localhost/operations/operations.asmx]
- nous utilisons le client tcp générique dans une fenêtre DOS

```

dos>clttcpgenerique localhost 80
Commandes :
POST /operations/operations.asmx/ajouter HTTP/1.1
HOST: localhost
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-length: 7

<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 13:55:17 GMT
<-- X-Powered-By: ASP.NET
<--
a=2&b=3
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 13:55:26 GMT
<-- X-Powered-By: ASP.NET
<-- Connection: close
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 90
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">5</double>
[fin du thread de lecture des réponses du serveur]
fin
[fin du thread d'envoi des commandes au serveur]

```

Remarquons tout d'abord que nous avons ajouté l'entête [Connection: close] pour demander au serveur de fermer la connexion après avoir envoyé la réponse. Cela est nécessaire ici. Si on ne le dit pas, par défaut le serveur va garder la connexion ouverte. Or sa réponse est une suite de lignes de texte dont la dernière n'est pas terminée par une marque de fin de ligne. Il se trouve que notre client TCP générique lit des lignes de texte terminées par la marque de fin de ligne avec la méthode *ReadLine*. Si le serveur ne ferme pas la connexion après envoi de la dernière ligne, le client est bloqué parce qu'il attend une marque de fin de ligne qui ne vient pas. Si le serveur ferme la connexion, la méthode *ReadLine* du client se termine et le client ne reste pas bloqué.

Aussitôt après avoir reçu la ligne vide signalant la fin des entêtes HTTP, le serveur IIS envoie une première réponse :

```

<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 13:55:17 GMT
<-- X-Powered-By: ASP.NET
<--

```

Cette réponse formée uniquement d'entêtes HTTP indique au client qu'il peut envoyer les 7 caractères qu'il a dit vouloir envoyer. Ce que nous faisons :

```
a=2&b=3
```

Il faut voir ici que notre client tcp envoie plus de 7 caractères puisqu'il les envoie avec une marque de fin de ligne (*WriteLine*). Ça ne gêne pas le serveur qui des caractères reçus ne prendra que les 7 premiers et parce qu'ensuite la connexion est fermée (*Connection: close*). Ces caractères en trop auraient été gênants si la connexion était restée ouverte car alors ils auraient été pris comme venant de la commande suivante du client. Une fois les paramètres reçus, le serveur envoie sa réponse :

```

<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 13:55:26 GMT
<-- X-Powered-By: ASP.NET
<-- Connection: close
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 90
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">5</double>

```

Nous avons maintenant les éléments pour écrire un client programmé pour notre service web. Ce sera un client console appelé **httpPost2** et s'utilisant comme suit :

```

dos>httpPost2 http://localhost/operations/operations.asmx
Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b

```

```

ajouter 6 7
--> POST /operations/operations.asmx/ajouter HTTP/1.1
--> Host: localhost:80
--> Content-Type: application/x-www-form-urlencoded
--> Content-Length: 7
--> Connection: Keep-Alive
-->
<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:38 GMT
<-- X-Powered-By: ASP.NET
<--
--> a=6&b=7
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:38 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 91
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">13</double>
[résultat=13]

soustraire 8 9
--> POST /operations/operations.asmx/soustraire HTTP/1.1
--> Host: localhost:80
--> Content-Type: application/x-www-form-urlencoded
--> Content-Length: 7
--> Connection: Keep-Alive
-->
<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:47 GMT
<-- X-Powered-By: ASP.NET
<--
--> a=8&b=9
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:47 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 91
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">-1</double>
[résultat=-1]

fin
dos>

```

Le client est appelé en lui passant l'URL du service web :

```
dos>httpPost2 http://localhost/operations/operations.asmx
```

Ensuite, le client lit les commandes tapées au clavier et les exécute. Celles-ci sont au format :

#### fonction a b

où **fonction** est la fonction du service web appelée (ajouter, soustraire, multiplier, diviser) et **a** et **b** les valeurs sur lesquelles va opérer cette fonction. Par exemple :

```
ajouter 6 7
```

A partir de là, le client va faire la requête HTTP nécessaire au serveur Web et obtenir une réponse. Les échanges client-serveur sont dupliqués à l'écran pour une meilleure compréhension du processus :

```

ajouter 6 7
--> POST /operations/operations.asmx/ajouter HTTP/1.1
--> Host: localhost:80
--> Content-Type: application/x-www-form-urlencoded
--> Content-Length: 7
--> Connection: Keep-Alive
-->

```

```

<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:38 GMT
<-- X-Powered-By: ASP.NET
<--
--> a=6&b=7
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:38 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 91
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">13</double>
[résultat=13]

```

On retrouve ci-dessus l'échange déjà rencontré avec le client tcp générique à une différence près : l'entête **HTTP Connection: Keep-Alive** demande au serveur de ne pas fermer la connexion. Celle-ci reste donc ouverte pour l'opération suivante du client qui n'a donc pas besoin de se reconnecter de nouveau au serveur. Cela l'oblige cependant à utiliser une autre méthode que *ReadLine()* pour lire la réponse du serveur puisqu'on sait que celle-ci est une suite de lignes dont la dernière n'est pas terminée par une marque de fin de ligne. Une fois toute la réponse du serveur obtenue, le client l'analyse pour y trouver le résultat de l'opération demandée et l'afficher :

```
[résultat=13]
```

Examinons le code de notre client :

```

' espaces de noms
Imports System
Imports System.Net.Sockets
Imports System.IO
Imports System.Text.RegularExpressions
Imports System.Collections
Imports Microsoft.VisualBasic
Imports System.Web

' client d'un service web operations
Public Module clientPOST

Public Sub Main(ByVal args() As String)
    ' syntaxe
    Const syntaxe As String = "pg URI"
    Dim fonctions As String() = {"ajouter", "soustraire", "multiplier", "diviser"}

    ' nombre d'arguments
    If args.Length <> 1 Then
        erreur(syntaxe, 1)
    End If

    ' on note l'URI demandée
    Dim URIstring As String = args(0)

    ' on se connecte au serveur
    Dim uri As Uri = Nothing ' l'URI du service web
    Dim client As TcpClient = Nothing ' la liaison tcp du client avec le serveur
    Dim [IN] As StreamReader = Nothing ' le flux de lecture du client
    Dim OUT As StreamWriter = Nothing ' le flux d'écriture du client
    Try
        ' connexion au serveur
        uri = New Uri(URIstring)
        client = New TcpClient(uri.Host, uri.Port)
        ' on crée les flux d'entrée-sortie du client TCP
        [IN] = New StreamReader(client.GetStream())
        OUT = New StreamWriter(client.GetStream())
        OUT.AutoFlush = True
    Catch ex As Exception
        ' URI incorrecte ou autre problème
        erreur("L'erreur suivante s'est produite : " + ex.Message, 2)
    End Try

    ' création d'un dictionnaire des fonctions du service web
    Dim dicoFonctions As New Hashtable
    Dim i As Integer
    For i = 0 To fonctions.Length - 1
        dicoFonctions.Add(fonctions(i), True)
    Next i

```

```

' les demandes de l'utilisateur sont tapées au clavier
' sous la forme fonction a b
' elles se terminent avec la commande fin
Dim commande As String = Nothing ' commande tapée au clavier
Dim champs As String() = Nothing ' champs d'une ligne de commande
Dim fonction As String = Nothing ' nom d'une fonction du service web
Dim a, b As String ' les arguments des fonctions du service web

' invite à l'utilisateur
Console.Out.WriteLine("Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b")

' gestion des erreurs
Dim erreurCommande As Boolean
Try
' boucle de saisie des commandes tapées au clavier
While True
' pas d'erreur au départ
erreurCommande = False
' lecture commande
commande = Console.In.ReadLine().Trim().ToLower()
' fini ?
If commande Is Nothing Or commande = "fin" Then
Exit While
End If
' décomposition de la commande en champs
champs = Regex.Split(commande, "\s+")
Try
' il faut trois champs
If champs.Length <> 3 Then
Throw New Exception
End If
' le champ 0 doit être une fonction reconnue
fonction = champs(0)
If Not dicoFonctions.ContainsKey(fonction) Then
Throw New Exception
End If
' le champ 1 doit être un nombre valide
a = champs(1)
Double.Parse(a)
' le champ 2 doit être un nombre valide
b = champs(2)
Double.Parse(b)
Catch
' commande invalide
Console.Out.WriteLine("syntaxe : [ajouter|soustraire|multiplier|diviser] a b")
erreurCommande = True
End Try
' on fait la demande au service web
If Not erreurCommande Then executeFonction([IN], OUT, uri, fonction, a, b)
End While
Catch e As Exception
Console.Out.WriteLine(("L'erreur suivante s'est produite : " + e.Message))
End Try
' fin liaison client-serveur
Try
[IN].Close()
OUT.Close()
client.Close()
Catch
End Try
End Sub
.....
' affichage des erreurs
Public Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
' affichage erreur
System.Console.Error.WriteLine(msg)
' arrêt avec erreur
Environment.Exit(exitCode)
End Sub
End Module

```

On a là des choses déjà rencontrées plusieurs fois et qui ne nécessitent pas de commentaires particuliers. Examinons maintenant le code de la méthode *executeFonction* où résident les nouveautés :

```

' executeFonction
Public Sub executeFonction(ByVal [IN] As StreamReader, ByVal OUT As StreamWriter, ByVal uri As Uri,
ByVal fonction As String, ByVal a As String, ByVal b As String)
' exécute fonction(a,b) sur le service web d'URI uri
' les échanges client-serveur se font via les flux IN et OUT
' le résultat de la fonction est dans la ligne

```

```

' <double xmlns="st.istia.univ-angers.fr">double</double>
' envoyée par le serveur

' construction de la chaîne de requête
Dim requête As String = "a=" + HttpUtility.UrlEncode(a) + "&b=" + HttpUtility.UrlEncode(b)
Dim nbChars As Integer = requête.Length

' construction du tableau des entêtes HTTP à envoyer
Dim entetes(5) As String
entetes(0) = "POST " + uri.AbsolutePath + "/" + fonction + " HTTP/1.1"
entetes(1) = "Host: " & uri.Host & ":" & uri.Port
entetes(2) = "Content-Type: application/x-www-form-urlencoded"
entetes(3) = "Content-Length: " & nbChars
entetes(4) = "Connection: Keep-Alive"
entetes(5) = ""

' on envoie les entêtes HTTP au serveur
Dim i As Integer
For i = 0 To entetes.Length - 1
    ' envoi au serveur
    OUT.WriteLine(entetes(i))
    ' écho écran
    Console.Out.WriteLine("--> " + entetes(i))
Next i

' on lit la lere réponse du serveur Web HTTP/1.1 100
Dim ligne As String = Nothing
' une ligne du flux de lecture
ligne = [IN].ReadLine()
While ligne <> ""
    'écho
    Console.Out.WriteLine("<-- " + ligne)
    ' ligne suivante
    ligne = [IN].ReadLine()
End While
'écho dernière ligne
Console.Out.WriteLine("<-- " + ligne)

' envoi paramètres de la requête
OUT.Write(requête)
' echo
Console.Out.WriteLine("--> " + requête))

' construction de l'expression régulière permettant de retrouver la taille de la réponse XML
' dans le flux de la réponse du serveur web
Dim modèleLength As String = "^Content-Length: (.+?)\s*$"
Dim RegexLength As New Regex(modèleLength)
Dim MatchLength As Match = Nothing
Dim longueur As Integer = 0

' lecture seconde réponse du serveur web après envoi de la requête
' on mémorise la valeur de la ligne Content-Length
ligne = [IN].ReadLine()
While ligne <> ""
    ' écho écran
    Console.Out.WriteLine("<-- " + ligne)
    ' Content-Length ?
    MatchLength = RegexLength.Match(ligne)
    If MatchLength.Success Then
        longueur = Integer.Parse(MatchLength.Groups(1).Value)
    End If
    ' ligne suivante
    ligne = [IN].ReadLine()
End While
' écho dernière ligne
Console.Out.WriteLine("<--")

' construction de l'expression régulière permettant de retrouver le résultat
' dans le flux de la réponse du serveur web
Dim modèle As String = "<double xmlns=""st.istia.univ-angers.fr"">(.*?)</double>"
Dim ModèleRésultat As New Regex(modèle)
Dim MatchRésultat As Match = Nothing

' on lit le reste de la réponse du serveur web
Dim chrRéponse(longueur) As Char
[IN].Read(chrRéponse, 0, longueur)
Dim strRéponse As String = New [String](chrRéponse)

' on décompose la réponse en lignes de texte
Dim lignes As String() = Regex.Split(strRéponse, ControlChars.Lf)

```

```

' on parcourt les lignes de texte à la recherche du résultat
Dim strRésultat As String = "?" ' résultat de la fonction
For i = 0 To lignes.Length - 1
    ' suivi
    Console.Out.WriteLine("<-- " + lignes(i))
    ' comparaison ligne courante au modèle
    MatchRésultat = ModèleRésultat.Match(lignes(i))
    ' a-t-on trouvé ?
    If MatchRésultat.Success Then
        ' on note le résultat
        strRésultat = MatchRésultat.Groups(1).Value
    End If
Next i
' on affiche le résultat
Console.Out.WriteLine("[résultat=" + strRésultat + "]" + ControlChars.Lf)
End Sub

```

Tout d'abord, le client HTTP-POST envoie sa demande au format POST :

```

' construction de la chaîne de requête
Dim requête As String = "a=" + HttpUtility.UrlEncode(a) + "&b=" + HttpUtility.UrlEncode(b)
Dim nbChars As Integer = requête.Length

' construction du tableau des entêtes HTTP à envoyer
Dim entetes(5) As String
entetes(0) = "POST " + uri.AbsolutePath + "/" + fonction + " HTTP/1.1"
entetes(1) = "Host: " & uri.Host & ":" & uri.Port
entetes(2) = "Content-Type: application/x-www-form-urlencoded"
entetes(3) = "Content-Length: " & nbChars
entetes(4) = "Connection: Keep-Alive"
entetes(5) = ""

' on envoie les entêtes HTTP au serveur
Dim i As Integer
For i = 0 To entetes.Length - 1
    ' envoi au serveur
    OUT.WriteLine(entetes(i))
    ' écho écran
    Console.Out.WriteLine("--> " + entetes(i))
Next i

```

Dans l'entête

```
--> Content-Length: 7
```

on doit indiquer la taille des paramètres qui seront envoyés par le client derrière les entêtes HTTP :

```
--> a=6&b=7
```

Pour cela on utilise le code suivant :

```

' construction de la chaîne de requête
Dim requête As String = "a=" + HttpUtility.UrlEncode(a) + "&b=" + HttpUtility.UrlEncode(b)
Dim nbChars As Integer = requête.Length

```

La méthode `HttpUtility.UrlEncode(string chaîne)` transforme certains des caractères de *chaîne* en `%n1n2` où `n1n2` est le code ASCII du caractère transformé. Les caractères visés par cette transformation sont tous les caractères ayant un sens particulier dans une requête POST (l'espace, le signe =, le signe &, ...). Ici la méthode `HttpUtility.UrlEncode` est normalement inutile puisque *a* et *b* sont des nombres qui ne contiennent aucun de ces caractères particuliers. Elle est ici employée à titre d'exemple. Elle a besoin de l'espace de noms `System.Web`. Une fois que le client a envoyé ses entêtes HTTP :

```

--> POST /operations/operations.asmx/ajouter HTTP/1.1
--> Host: localhost:80
--> Content-Type: application/x-www-form-urlencoded
--> Content-Length: 7
--> Connection: Keep-Alive
-->

```

le serveur répond par l'entête HTTP `100 Continue` :

```

<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:47 GMT
<-- X-Powered-By: ASP.NET
<--

```

Le code se contente de lire et d'afficher à l'écran cette première réponse :

```
' on lit la 1ere réponse du serveur Web HTTP/1.1 100
Dim ligne As String = Nothing
' une ligne du flux de lecture
ligne = [IN].ReadLine()
While ligne <> ""
    'écho
    Console.Out.WriteLine("<!-- " + ligne)
    ' ligne suivante
    ligne = [IN].ReadLine()
End While
'écho dernière ligne
Console.Out.WriteLine("<!-- " + ligne)
```

Une fois cette première réponse lue, le client doit envoyer ses paramètres :

```
--> a=6&b=7
```

Il le fait avec le code suivant :

```
' envoi paramètres de la requête
OUT.Write(requête)
' echo
Console.Out.WriteLine("--> " + requête)
```

Le serveur va alors envoyer sa réponse. Celle-ci est composée de deux parties :

1. des entêtes HTTP terminés par une ligne vide
2. la réponse au format XML

```
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Wed, 03 Mar 2004 14:56:38 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 91
<--
<-- <?xml version="1.0" encoding="utf-8"?>
<-- <double xmlns="st.istia.univ-angers.fr">13</double>
```

Dans un premier temps, le client lit les entêtes HTTP pour y trouver la ligne *Content-Length* et récupérer la taille de la réponse XML (ici 90). Celle-ci est récupérée au moyen d'une expression régulière. On aurait pu faire autrement et sans doute de façon plus efficace.

```
' construction de l'expression régulière permettant de retrouver la taille de la réponse XML
' dans le flux de la réponse du serveur web
Dim modèleLength As String = "^Content-Length: (.+?)\s*$"
Dim RegexLength As New Regex(modèleLength)
Dim MatchLength As Match = Nothing
Dim longueur As Integer = 0

' lecture seconde réponse du serveur web après envoi de la requête
' on mémorise la valeur de la ligne Content-Length
ligne = [IN].ReadLine()
While ligne <> ""
    ' écho écran
    Console.Out.WriteLine("<!-- " + ligne)
    ' Content-Length ?
    MatchLength = RegexLength.Match(ligne)
    If MatchLength.Success Then
        longueur = Integer.Parse(MatchLength.Groups(1).Value)
    End If
    ' ligne suivante
    ligne = [IN].ReadLine()
End While
' écho dernière ligne
Console.Out.WriteLine("<!--")
```

Une fois qu'on a la longueur N de la réponse XML, on n'a plus qu'à lire N caractères dans le flux IN de la réponse du serveur. Cette chaîne de N caractères est redécomposée en lignes de texte pour les besoins du suivi écran. Parmi ces lignes on cherche la ligne du résultat :

```
<-- <double xmlns="st.istia.univ-angers.fr">13</double>
```

au moyen là encore d'une expression régulière. Une fois le résultat trouvé, il est affiché.

```
[résultat=13]
```

La fin du code du client est la suivante :

```
' construction de l'expression régulière permettant de retrouver le résultat
' dans le flux de la réponse du serveur web
Dim modèle As String = "<double xmlns="st.istia.univ-angers.fr">(.*?)</double>"
Dim ModèleRésultat As New Regex(modèle)
Dim MatchRésultat As Match = Nothing

' on lit le reste de la réponse du serveur web
Dim chrRéponse(longueur) As Char
[IN].Read(chrRéponse, 0, longueur)
Dim strRéponse As String = New [String](chrRéponse)

' on décompose la réponse en lignes de texte
Dim lignes As String() = Regex.Split(strRéponse, ControlChars.Lf)

' on parcourt les lignes de texte à la recherche du résultat
Dim strRésultat As String = "?" ' résultat de la fonction
For i = 0 To lignes.Length - 1
    ' suivi
    Console.Out.WriteLine("<!-- " + lignes(i))
    ' comparaison ligne courante au modèle
    MatchRésultat = ModèleRésultat.Match(lignes(i))
    ' a-t-on trouvé ?
    If MatchRésultat.Success Then
        ' on note le résultat
        strRésultat = MatchRésultat.Groups(1).Value
    End If
Next i
' on affiche le résultat
Console.Out.WriteLine("[résultat=" + strRésultat + "]" + ControlChars.Lf)
End Sub
```

## 9.6 Un client SOAP

Nous étudions ici un second client qui va lui utiliser un dialogue client-serveur de type **SOAP** (Simple Object Access Protocol). Un exemple de dialogue nous est présenté pour la fonction *ajouter* :

Adresse  OK

### SOAP

Le texte suivant est un exemple de demande et de réponse SOAP. Les **espaces réservés** affichés doivent être remplacés par des valeurs réelles.

```
POST /operations/operations.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "st.istia.univ-angers.fr/ajouter"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.o
  <soap:Body>
    <ajouter xmlns="st.istia.univ-angers.fr">
      <a>double</a>
      <b>double</b>
    </ajouter>
  </soap:Body>
</soap:Envelope>
```

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org
  <soap:Body>
    <ajouterResponse xmlns="st.istia.univ-angers.fr">
      <ajouterResult>double</ajouterResult>
    </ajouterResponse>
  </soap:Body>
</soap:Envelope>

```

La demande du client est une demande POST. On va donc retrouver certains des mécanismes du client précédent. La principale différence est qu'alors que le client HTTP-POST envoyait les paramètres *a* et *b* sous la forme

$$a=A&b=B$$

le client SOAP les envoie dans un format XML plus complexe :

```

POST /operations/operations.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "st.istia.univ-angers.fr/ajouter"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ajouter xmlns="st.istia.univ-angers.fr">
      <a>double</a>
      <b>double</b>
    </ajouter>
  </soap:Body>
</soap:Envelope>

```

Il reçoit en retour une réponse XML également plus complexe que les réponses vues précédemment :

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ajouterResponse xmlns="st.istia.univ-angers.fr">
      <ajouterResult>double</ajouterResult>
    </ajouterResponse>
  </soap:Body>
</soap:Envelope>

```

Même si la demande et la réponse sont plus complexes, il s'agit bien du même mécanisme HTTP que pour le client HTTP-POST. L'écriture du client SOAP peut être ainsi calquée sur celle du client HTTP-POST. Voici un exemple d'exécution :

```

dos>clientsoap1 http://localhost/operations/operations.asmx
Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b

ajouter 3 4
--> POST /operations/operations.asmx HTTP/1.1
--> Host: localhost:80
--> Content-Type: text/xml; charset=utf-8
--> Content-Length: 321
--> Connection: Keep-Alive
--> SOAPAction: "st.istia.univ-angers.fr/ajouter"
-->
<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 07:28:29 GMT
<-- X-Powered-By: ASP.NET
<--
--> <?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>

```

```

<ajouter xmlns="st.istia.univ-angers.fr">
<a>3</a>
<b>4</b>
</ajouter>
</soap:Body>
</soap:Envelope>
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 07:28:33 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 345
<--
<-- <?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
ance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><ajouterResponse xmlns="st.istia.univ-
angers.fr"><ajouterResult>7</ajouterResult></ajouterResponse
></soap:Body></soap:Envelope>
[résultat=7]

```

Seule la méthode *executeFonction* change. Le client SOAP envoie les entêtes HTTP de sa demande. Ils sont simplement un peu plus complexes que ceux de HTTP-POST :

```

ajouter 3 4
--> POST /operations/operations.asmx HTTP/1.1
--> Host: localhost:80
--> Content-Type: text/xml; charset=utf-8
--> Content-Length: 321
--> Connection: Keep-Alive
--> SOAPAction: "st.istia.univ-angers.fr/ajouter"
-->

```

Le code qui les génère :

```

' executeFonction
Public Sub executeFonction(ByVal [IN] As StreamReader, ByVal OUT As StreamWriter, ByVal uri As Uri,
ByVal fonction As String, ByVal a As String, ByVal b As String)
' exécute fonction(a,b) sur le service web d'URI uri
' les échanges client-serveur se font via les flux IN et OUT
' le résultat de la fonction est dans la ligne
' <double xmlns="st.istia.univ-angers.fr">double</double>
' envoyée par le serveur
' construction de la chaîne de requête SOAP

Dim requêteSOAP As String = "<?xml version=" + ""1.0"" encoding=""utf-8""?>" + ControlChars.Lf
requêteSOAP += "<soap:Envelope xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
xmlns:xsd=""http://www.w3.org/2001/XMLSchema""
xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"">" + ControlChars.Lf
requêteSOAP += "<soap:Body>" + ControlChars.Lf
requêteSOAP += "<" + fonction + " xmlns=""st.istia.univ-angers.fr"">" + ControlChars.Lf
requêteSOAP += "<a>" + a + "</a>" + ControlChars.Lf
requêteSOAP += "<b>" + b + "</b>" + ControlChars.Lf
requêteSOAP += "</" + fonction + ">" + ControlChars.Lf
requêteSOAP += "</soap:Body>" + ControlChars.Lf
requêteSOAP += "</soap:Envelope>"
Dim nbCharsSOAP As Integer = requêteSOAP.Length

' construction du tableau des entêtes HTTP à envoyer
Dim entetes(6) As String
entetes(0) = "POST " + uri.AbsolutePath + " HTTP/1.1"
entetes(1) = "Host: " & uri.Host & ":" & uri.Port
entetes(2) = "Content-Type: text/xml; charset=utf-8"
entetes(3) = "Content-Length: " & nbCharsSOAP
entetes(4) = "Connection: Keep-Alive"
entetes(5) = "SOAPAction: ""st.istia.univ-angers.fr/" + fonction + """"
entetes(6) = ""

' on envoie les entêtes HTTP au serveur
Dim i As Integer
For i = 0 To entetes.Length - 1
' envoi au serveur
OUT.WriteLine(entetes(i))
' écho écran
Console.Out.WriteLine("--> " + entetes(i))
Next i

```

En recevant cette demande, le serveur envoie sa première réponse que le client affiche :

```
<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 07:28:29 GMT
<-- X-Powered-By: ASP.NET
<--
```

Le code de lecture de cette première réponse est le suivant :

```
' on lit la 1ere réponse du serveur Web HTTP/1.1 100
Dim ligne As String = Nothing
' une ligne du flux de lecture
ligne = [IN].ReadLine()
While ligne <> ""
    'écho
    Console.Out.WriteLine("<-- " + ligne)
    ' ligne suivante
    ligne = [IN].ReadLine()
End While 'while
'écho dernière ligne
Console.Out.WriteLine("<-- " + ligne)
```

Le client va maintenant envoyer ses paramètres au format XML dans quelque chose qu'on appelle une enveloppe SOAP :

```
--> <?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ajouter xmlns="st.istia.univ-angers.fr">
<a>3</a>
<b>4</b>
</ajouter>
</soap:Body>
</soap:Envelope>
```

Le code :

```
' envoi paramètres de la requête
OUT.Write(requêteSOAP)
' echo
Console.Out.WriteLine("--> " + requêteSOAP)
```

Le serveur va alors envoyer sa réponse définitive :

```
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 07:28:33 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 345
<--
<-- <?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
ance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><ajouterResponse xmlns="st.istia.univ-
angers.fr"><ajouterResult>7</ajouterResult></ajouterResponse
></soap:Body></soap:Envelope>
```

Le client affiche à l'écran les entêtes HTTP reçus tout en y cherchant la ligne *Content-Length* :

```
' construction de l'expression régulière permettant de retrouver la taille de la réponse XML
' dans le flux de la réponse du serveur web
Dim modèleLength As String = "^Content-Length: (.+?)\s*$"
Dim RegexLength As New Regex(modèleLength)
Dim MatchLength As Match = Nothing
Dim longueur As Integer = 0

' lecture seconde réponse du serveur web après envoi de la requête
' on mémorise la valeur de la ligne Content-Length
ligne = [IN].ReadLine()
While ligne <> ""
    ' écho écran
    Console.Out.WriteLine("<-- " + ligne)
    ' Content-Length ?
    MatchLength = RegexLength.Match(ligne)
    If MatchLength.Success Then
        longueur = Integer.Parse(MatchLength.Groups(1).Value)
```

```

End If
' ligne suivante
ligne = [IN].ReadLine()
End While 'while
' écho dernière ligne
Console.Out.WriteLine("<!--")

```

Une fois la taille N de la réponse XML connue, le client lit N caractères dans le flux de la réponse du serveur, décompose la chaîne récupérée en lignes de texte pour les afficher à l'écran et y chercher la balise XML du résultat : `<ajouterResult>?</ajouterResult>` et afficher ce dernier :

```

' construction de l'expression régulière permettant de retrouver le résultat
' dans le flux de la réponse du serveur web
Dim modèle As String = "<" + fonction + "Result>(.*?)</" + fonction + "Result>"
Dim ModèleRésultat As New Regex(modèle)
Dim MatchRésultat As Match = Nothing

' on lit le reste de la réponse du serveur web
Dim chrRéponse(longueur) As Char
[IN].Read(chrRéponse, 0, longueur)
Dim strRéponse As String = New [String](chrRéponse)

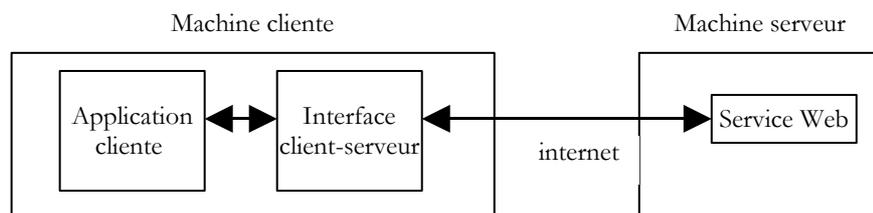
' on décompose la réponse en lignes de texte
Dim lignes As String() = Regex.Split(strRéponse, ControlChars.Lf)

' on parcourt les lignes de texte à la recherche du résultat
Dim strRésultat As String = "?" ' résultat de la fonction
For i = 0 To lignes.Length - 1
' suivi
Console.Out.WriteLine("<!-- " + lignes(i))
' comparaison ligne courante au modèle
MatchRésultat = ModèleRésultat.Match(lignes(i))
' a-t-on trouvé ?
If MatchRésultat.Success Then
' on note le résultat
strRésultat = MatchRésultat.Groups(1).Value
End If
' ligne suivante
Next i
' on affiche le résultat
Console.Out.WriteLine("[résultat=" + strRésultat + "]" + ControlChars.Lf)
End Sub

```

## 9.7 Encapsulation des échanges client-serveur

Imaginons que notre service web *operations* soit utilisé par diverses applications. Il serait intéressant de mettre à disposition de celles-ci une classe qui ferait l'interface entre l'application cliente et le service web et qui cacherait la majeure partie des échanges réseau qui, pour la plupart des développeurs, ne sont pas triviaux. On aurait ainsi le schéma suivant :



L'application cliente s'adresserait à l'interface client-serveur pour faire ses demandes au service web. Celle-ci ferait tous les échanges réseau nécessaires avec le serveur et rendrait le résultat obtenu à l'application cliente. Celle-ci n'aurait plus à s'occuper des échanges avec le serveur ce qui faciliterait grandement son écriture.

### 9.7.1 La classe d'encapsulation

Après ce qui a été vu dans les paragraphes précédents, nous connaissons bien maintenant les échanges réseau entre le client et le serveur. Nous avons même vu trois méthodes. Nous choisissons d'encapsuler la méthode SOAP. La classe est la suivante :

```

' espaces de noms
Imports System
Imports System.Net.Sockets
Imports System.IO
Imports System.Text.RegularExpressions

```

```

Imports System.Collections
Imports System.Web
Imports Microsoft.VisualBasic

' clientSOAP du service Web operations
Public Class clientSOAP

' variables d'instance
Private uri As Uri = Nothing ' l'URI du service web
Private client As TcpClient = Nothing ' la liaison tcp du client avec le serveur
Private [IN] As StreamReader = Nothing ' le flux de lecture du client
Private OUT As StreamWriter = Nothing ' le flux d'écriture du client
' dictionnaire des fonctions
Private dicoFonctions As New Hashtable
' liste des fonctions
Private fonctions As String() = {"ajouter", "soustraire", "multiplier", "diviser"}
' verbose
Private verbose As Boolean = False ' à vrai, affiche à l'écran les échanges client-serveur

' constructeur
Public Sub New(ByVal uriString As String, ByVal verbose As Boolean)

' on note verbose
Me.verbose = verbose

' connexion au serveur
uri = New Uri(uriString)
client = New TcpClient(uri.Host, uri.Port)

' on crée les flux d'entrée-sortie du client TCP
[IN] = New StreamReader(client.GetStream())
OUT = New StreamWriter(client.GetStream())
OUT.AutoFlush = True

' création du dictionnaire des fonctions du service web
Dim i As Integer
For i = 0 To fonctions.Length - 1
    dicoFonctions.Add(fonctions(i), True)
Next i
End Sub

' fermeture de la connexion au serveur
Public Sub Close()
' fin liaison client-serveur
[IN].Close()
OUT.Close()
client.Close()
End Sub

' executeFonction
Public Function executeFonction(ByVal fonction As String, ByVal a As String, ByVal b As String) As
String
' exécute fonction(a,b) sur le service web d'URI uri
' les échanges client-serveur se font via les flux IN et OUT
' le résultat de la fonction est dans la ligne
' <double xmlns="st.istia.univ-angers.fr">double</double>
' envoyée par le serveur

' fonction valide ?
fonction = fonction.Trim().ToLower()
If Not dicoFonctions.ContainsKey(fonction) Then
    Return "[fonction [" + fonction + "] indisponible : (ajouter, soustraire,multiplier,diviser)]"
End If

' arguments a et b valides ?
Dim doubleA As Double = 0
Try
    doubleA = Double.Parse(a)
Catch
    Return "[argument [" + a + "] incorrect (double)]"
End Try
Dim doubleB As Double = 0
Try
    doubleB = Double.Parse(b)
Catch
    Return "[argument [" + b + "] incorrect (double)]"
End Try

' division par zéro ?
If fonction = "diviser" And doubleB = 0 Then
    Return "[division par zéro]"

```

```

End If

' construction de la chaîne de requête SOAP
Dim requêteSOAP As String = "<?xml version=" + """"1.0"" encoding=""utf-8""?>" + ControlChars.Lf
requêteSOAP += "<soap:Envelope xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance""
xmlns:xsd=""http://www.w3.org/2001/XMLSchema""
xmlns:soap=""http://schemas.xmlsoap.org/soap/envelope/"">" + ControlChars.Lf
requêteSOAP += "<soap:Body>" + ControlChars.Lf
requêteSOAP += "<" + fonction + " xmlns=""st.istia.univ-angers.fr"">" + ControlChars.Lf
requêteSOAP += "<a>" + a + "</a>" + ControlChars.Lf
requêteSOAP += "<b>" + b + "</b>" + ControlChars.Lf
requêteSOAP += "</" + fonction + ">" + ControlChars.Lf
requêteSOAP += "</soap:Body>" + ControlChars.Lf
requêteSOAP += "</soap:Envelope>"
Dim nbCharsSOAP As Integer = requêteSOAP.Length

' construction du tableau des entêtes HTTP à envoyer
Dim entetes(6) As String
entetes(0) = "POST " + uri.AbsolutePath + " HTTP/1.1"
entetes(1) = "Host: " + uri.Host + ":" + uri.Port.ToString
entetes(2) = "Content-Type: text/xml; charset=utf-8"
entetes(3) = "Content-Length: " + nbCharsSOAP.ToString
entetes(4) = "Connection: Keep-Alive"
entetes(5) = "SOAPAction: ""st.istia.univ-angers.fr/" + fonction + """"
entetes(6) = ""

' on envoie les entêtes HTTP au serveur
Dim i As Integer
For i = 0 To entetes.Length - 1
    ' envoi au serveur
    OUT.WriteLine(entetes(i))
    ' écho écran
    If verbose Then
        Console.Out.WriteLine(("<-- " + entetes(i)))
    End If
Next i

' on lit la 1ere réponse du serveur Web HTTP/1.1 100
Dim ligne As String = Nothing
' une ligne du flux de lecture
ligne = [IN].ReadLine()
While ligne <> ""
    'écho
    If verbose Then
        Console.Out.WriteLine(("<-- " + ligne))
    End If
    ' ligne suivante
    ligne = [IN].ReadLine()
End While
'écho dernière ligne
If verbose Then
    Console.Out.WriteLine(("<-- " + ligne))
End If
' envoi paramètres de la requête
OUT.Write(requêteSOAP)
' echo
If verbose Then
    Console.Out.WriteLine(("<--> " + requêteSOAP))
End If

' construction de l'expression régulière permettant de retrouver la taille de la réponse XML
' dans le flux de la réponse du serveur web
Dim modèleLength As String = "^Content-Length: (.+)\s*$"
Dim RegexLength As New Regex(modèleLength)
Dim MatchLength As Match = Nothing
Dim longueur As Integer = 0

' lecture seconde réponse du serveur web après envoi de la requête
' on mémorise la valeur de la ligne Content-Length
ligne = [IN].ReadLine()
While ligne <> ""
    ' écho écran
    If verbose Then
        Console.Out.WriteLine(("<-- " + ligne))
    End If
    ' Content-Length ?
    MatchLength = RegexLength.Match(ligne)
    If MatchLength.Success Then
        longueur = Integer.Parse(MatchLength.Groups(1).Value)
    End If
    ' ligne suivante

```

```

    ligne = [IN].ReadLine()
End While

' écho dernière ligne
If verbose Then
    Console.Out.WriteLine("<!--")
End If

' construction de l'expression régulière permettant de retrouver le résultat
' dans le flux de la réponse du serveur web
Dim modèle As String = "<" + fonction + "Result>(.*?)</" + fonction + "Result>"
Dim ModèleRésultat As New Regex(modèle)
Dim MatchRésultat As Match = Nothing

' on lit le reste de la réponse du serveur web
Dim chrRéponse(longueur) As Char
[IN].Read(chrRéponse, 0, longueur)
Dim strRéponse As String = New [String](chrRéponse)

' on décompose la réponse en lignes de texte
Dim lignes As String() = Regex.Split(strRéponse, ControlChars.Lf)

' on parcourt les lignes de texte à la recherche du résultat
Dim strRésultat As String = "?" ' résultat de la fonction
For i = 0 To lignes.Length - 1
    ' suivi
    If verbose Then
        Console.Out.WriteLine(("<-- " + lignes(i)))
    End If
    ' comparaison ligne courante au modèle
    MatchRésultat = ModèleRésultat.Match(lignes(i))
    ' a-t-on trouvé ?
    If MatchRésultat.Success Then
        ' on note le résultat
        strRésultat = MatchRésultat.Groups(1).Value
    End If
Next i

' on renvoie le résultat
Return strRésultat
End Function
End Class

```

Nous ne retrouvons rien de neuf par rapport à ce qui a été déjà vu. Nous avons simplement repris le code du client SOAP étudié et l'avons réaménagé quelque peu pour en faire une classe. Celle-ci a un constructeur et deux méthodes :

```

' constructeur
Public Sub New(ByVal uriString As String, ByVal verbose As Boolean)

' executeFonction
Public Function executeFonction(ByVal fonction As String, ByVal a As String, ByVal b As String) As String

' fermeture de la connexion au serveur
Public Sub Close()

```

et a les attributs suivants :

```

' variables d'instance
Private uri As Uri = Nothing ' l'URI du service web
Private client As TcpClient = Nothing ' la liaison tcp du client avec le serveur
Private [IN] As StreamReader = Nothing ' le flux de lecture du client
Private OUT As StreamWriter = Nothing ' le flux d'écriture du client
' dictionnaire des fonctions
Private dicoFonctions As New Hashtable
' liste des fonctions
Private fonctions As String() = {"ajouter", "soustraire", "multiplier", "diviser"}
' verbose
Private verbose As Boolean = False ' à vrai, affiche à l'écran les échanges client-serveur

```

On passe au constructeur deux paramètres :

1. l'URI du service web auquel il doit se connecter
2. un booléen *verbose* qui, à vrai, demande que les échanges réseau soient affichés à l'écran, sinon ils ne le seront pas.

Au cours de la construction, on construit les flux *IN* de lecture réseau, *OUT* d'écriture réseau, ainsi que le dictionnaire des fonctions gérées par le service. Une fois l'objet construit, la connexion client-serveur est ouverte et ses flux *IN* et *OUT* utilisables.

La méthode *Close* permet de fermer la liaison avec le serveur.

La méthode *ExecuteFonction* est celle qu'on a écrite pour le client SOAP étudié, à quelques détails près :

1. Les paramètres uri, IN et OUT qui étaient auparavant passés en paramètres à la méthode n'ont plus besoin de l'être, puisque ce sont maintenant des attributs d'instance accessibles à toutes les méthodes de l'instance
2. La méthode *ExecuteFonction* qui rendait auparavant un type *void* et affichait le résultat de la fonction à l'écran, rend maintenant ce résultat et donc un type *string*.

Typiquement un client utilisera la classe *clientSOAP* de la façon suivante :

1. création d'un objet *clientSOAP* qui va créer la liaison avec le service web
2. utilisation répétée de la méthode *executeFonction*
3. fermeture de la liaison avec le service Web par la méthode *Close*.

Etudions un premier client.

## 9.7.2 Un client console

Nous reprenons ici le client SOAP étudié alors que la classe *clientSOAP* n'existait pas et nous le réaménageons afin qu'il utilise maintenant cette classe :

```
' espaces de noms
Imports System
Imports System.IO
Imports System.Text.RegularExpressions
Imports Microsoft.VisualBasic

Public Module testClientSoap

    ' demande l'URI du service web operations
    ' exécute de façon interactive les commandes tapées au clavier
    Public Sub Main(ByVal args() As String)
        ' syntaxe
        Const syntaxe As String = "pg URI [verbose]"

        ' nombre d'arguments
        If args.Length <> 1 And args.Length <> 2 Then
            erreur(syntaxe, 1)
        End If

        ' verbose ?
        Dim verbose As Boolean = False
        If args.Length = 2 Then
            verbose = args(1).ToLower() = "verbose"
        End If

        ' on se connecte au service web
        Dim client As clientSOAP = Nothing
        Try
            client = New clientSOAP(args(0), verbose)
        Catch ex As Exception
            ' erreur de connexion
            erreur("L'erreur suivante s'est produite lors de la connexion au service web : " + ex.Message, 2)
        End Try

        ' les demandes de l'utilisateur sont tapées au clavier
        ' sous la forme fonction a b - elles se terminent avec la commande fin
        Dim commande As String = Nothing      ' commande tapée au clavier
        Dim champs As String() = Nothing     ' champs d'une ligne de commande

        ' invite à l'utilisateur
        Console.Out.WriteLine("Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b" +
            ControlChars.Lf)

        ' gestion des erreurs
        Dim erreurCommande As Boolean
        Try
            ' boucle de saisie des commandes tapées au clavier
            While True
                ' au départ pas d'erreur
                erreurCommande = False
                ' lecture commande
                commande = Console.In.ReadLine().Trim().ToLower()
                ' fini ?
                If commande Is Nothing Or commande = "fin" Then
                    Exit While
                End If
            End While
        End Try
    End Sub
End Module
```

```

' décomposition de la commande en champs
champs = Regex.Split(commande, "\s+")
' il faut trois champs
If champs.Length <> 3 Then
    Console.Out.WriteLine("syntaxe : [ajouter|soustraire|multiplier|diviser] a b")
    ' on note l'erreur
    erreurCommande = True
End If
' on fait la demande au service web
If Not erreurCommande Then Console.Out.WriteLine("résultat=" +
client.executeFonction(champs(0).Trim().ToLower(), champs(1).Trim(), champs(2).Trim()))
' demande suivante
End While
Catch e As Exception
    Console.Out.WriteLine("L'erreur suivante s'est produite : " + e.Message)
End Try
' fin liaison client-serveur
Try
    client.Close()
Catch
End Try
End Sub

' affichage des erreurs
Public Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
    ' affichage erreur
    System.Console.Error.WriteLine(msg)
    ' arrêt avec erreur
    Environment.Exit(exitCode)
End Sub
End Module

```

La client est maintenant considérablement plus simple et on n'y retrouve aucune communication réseau. Le client admet deux paramètres :

1. l'URI du service web *operations*
2. le mot clé facultatif *verbose*. S'il est présent, les échanges réseau seront affichés à l'écran.

Ces deux paramètres sont utilisés pour construire un objet *clientSOAP* qui va assurer les échanges avec le service web.

```

' on se connecte au service web
Dim client As clientSOAP = Nothing
Try
    client = New clientSOAP(args(0), verbose)
Catch ex As Exception
    ' erreur de connexion
    erreur("L'erreur suivante s'est produite lors de la connexion au service web : " + ex.Message, 2)
End Try

```

Une fois ouverte la connexion avec le service web, le client peut envoyer ses demandes. Celles-ci sont tapées au clavier, analysées puis envoyées au serveur par appel à la méthode *executeFonction* de l'objet *clientSOAP*.

```

' on fait la demande au service web
If Not erreurCommande Then Console.Out.WriteLine("résultat=" +
client.executeFonction(champs(0).Trim().ToLower(), champs(1).Trim(), champs(2).Trim()))

```

La classe *clientSOAP* est compilée dans un "assemblage" :

```

dos>vb /r:clientSOAP.dll testClientSOAP.vb
dos>dir
04/03/2004 08:46          6 913 clientSOAP.vb
04/03/2004 09:07          7 168 clientSOAP.dll

```

L'application client *testClientSoap* est ensuite compilée par :

```

dos>vb /r:clientSOAP.dll /r:system.dll testClientSOAP.vb
dos>dir
04/03/2004 09:08          2 711 testClientSOAP.vb
04/03/2004 09:08          4 608 testClientSOAP.exe

```

Voici un exemple d'exécution non verbeux :

```

dos>testclientsoap http://localhost/st/operations/operations.asmx
Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b
ajouter 1 3
résultat=4

```

```

soustraire 6 7
résultat=-1
multiplier 4 5
résultat=20
diviser 1 2
résultat=0.5
x
syntaxe : [ajouter|soustraire|multiplier|diviser] a b
x 1 2
résultat=[fonction [x] indisponible : (ajouter, soustraire,multiplier,diviser)]
ajouter a b
résultat=[argument [a] incorrect (double)]
ajouter 1 b
résultat=[argument [b] incorrect (double)]
diviser 1 0
résultat=[division par zéro]
fin

```

On peut suivre les échanges réseau en demandant une exécutions "verbose" :

```

dos>testClientSOAP http://localhost/operations/operations.asmx verbose
Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser] a b

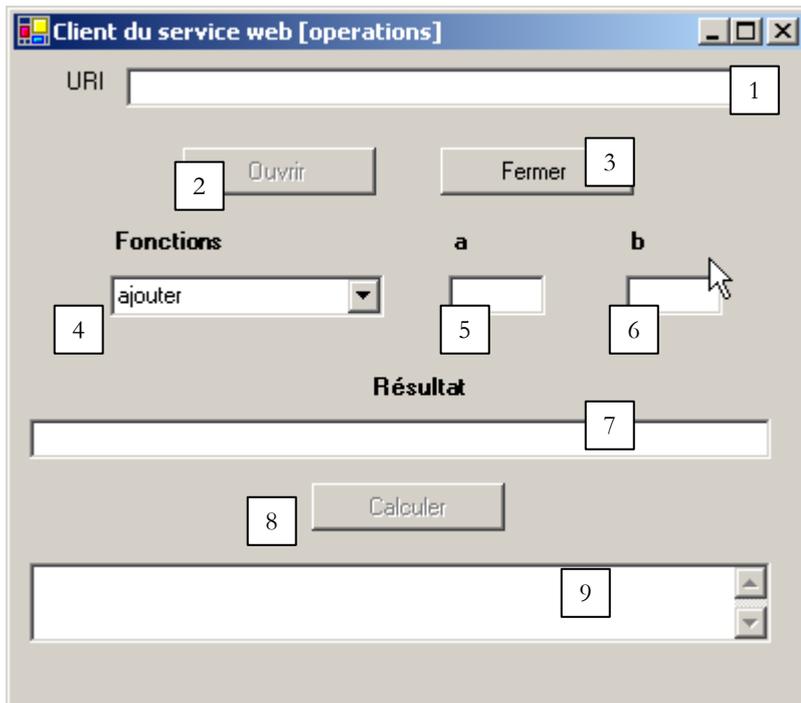
ajouter 4 8
--> POST /operations/operations.asmx HTTP/1.1
--> Host: localhost:80
--> Content-Type: text/xml; charset=utf-8
--> Content-Length: 321
--> Connection: Keep-Alive
--> SOAPAction: "st.istia.univ-angers.fr/ajouter"
-->
<-- HTTP/1.1 100 Continue
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 08:15:25 GMT
<-- X-Powered-By: ASP.NET
<--
--> <?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<ajouter xmlns="st.istia.univ-angers.fr">
<a>4</a>
<b>8</b>
</ajouter>
</soap:Body>
</soap:Envelope>
<-- HTTP/1.1 200 OK
<-- Server: Microsoft-IIS/5.0
<-- Date: Thu, 04 Mar 2004 08:15:25 GMT
<-- X-Powered-By: ASP.NET
<-- X-AspNet-Version: 1.1.4322
<-- Cache-Control: private, max-age=0
<-- Content-Type: text/xml; charset=utf-8
<-- Content-Length: 346
<--
<-- <?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
ance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><ajouterResponse xmlns="st.istia.univ-
angers.fr"><ajouterResult>12</ajouterResult></ajouterResponse></soap:Body></soap:Envelope>
résultat=12
fin

```

Construisons maintenant un client graphique.

## 9.7.3 Un client graphique windows

Nous allons maintenant interroger notre service web avec un client graphique qui utilisera lui aussi la classe *clientSOAP*. L'interface graphique sera la suivante :



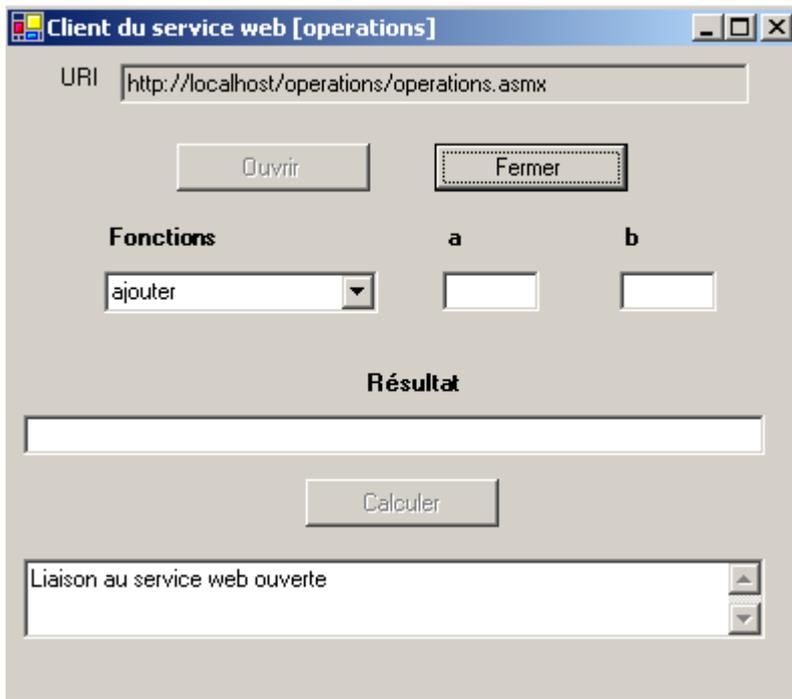
Les contrôles sont les suivants :

n°	type	nom	rôle
1	TextBox	txtURI	l'URI du service web operations
2	Button	btnOuvrir	ouvre la liaison avec le service Web
3	Button	btnFermer	ferme la liaison avec le service Web
4	ComboBox	cmbFonctions	la liste des fonction (ajouter, soustraire, multiplier, diviser)
5	TextBox	txtA	l'argument a des fonctions
6	TextBox	txtB	l'argument b des fonctions
7	TextBox	txtRésultat	le résultat de fonction(a,b)
8	Button	btnCalculer	lance le calcul de fonction(a,b)
9	TextBox	txtErreur	affiche un msg d'état de la liaison

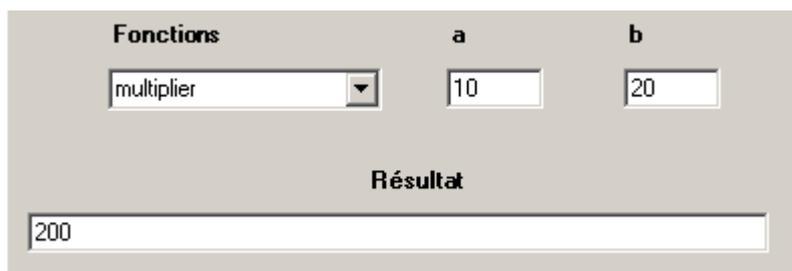
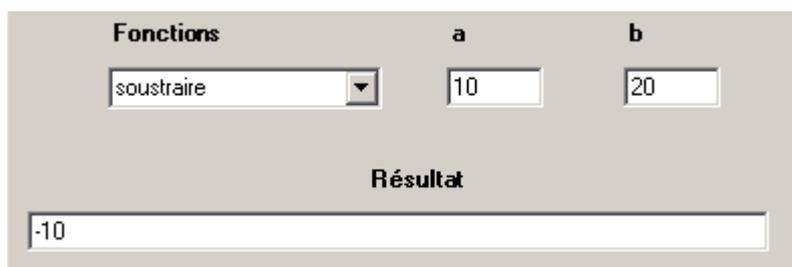
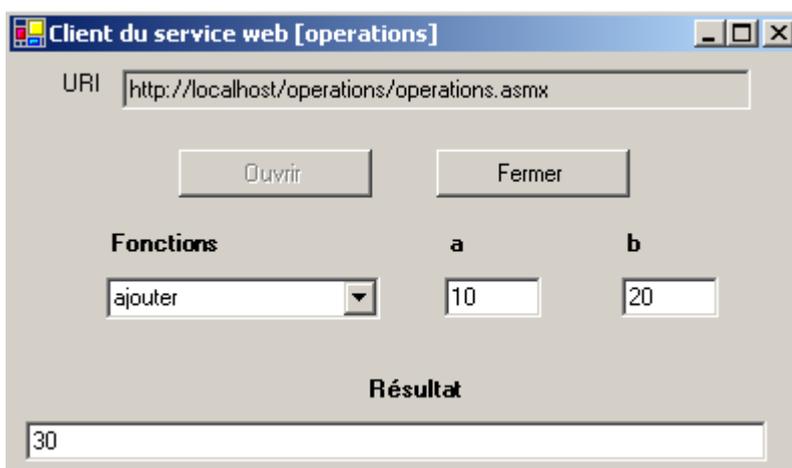
Il y a quelques contraintes de fonctionnement :

- le bouton *btnOuvrir* n'est actif que si le champ *txtURI* est non vide et qu'une liaison n'est pas déjà ouverte
- le bouton *btnFermer* n'est actif que lorsqu'une liaison avec le service web a été ouverte
- le bouton *btnCalculer* n'est actif que lorsqu'une liaison est ouverte et que les champs *txtA* et *txtB* sont non vides
- les champs *txtRésultat* et *txtErreur* ont l'attribut *ReadOnly* à vrai

Le client commence par ouvrir la connexion avec le service web à l'aide du bouton [Ouvrir] :



Ensuite l'utilisateur peut choisir une fonction et des valeurs pour a et b :



**Fonctions**                      **a**                      **b**

diviser                      10                      20

**Résultat**

0.5

**Fonctions**                      **a**                      **b**

diviser                      10                      0

**Résultat**

[division par zéro]

Le code de l'application suit. Nous avons omis le code du formulaire qui ne présente pas d'intérêt ici.

```
'espaces de noms
Imports System
Imports System.Windows.Forms

' la classe du formulaire
Public Class FormClientSOAP
    Inherits System.Windows.Forms.Form

    ' attributs d'instance
    Dim client As clientSOAP ' client SOAP du service web operations

#Region " Code généré par le Concepteur Windows Form "

    Public Sub New()
        MyBase.New()
        'Cet appel est requis par le Concepteur Windows Form.
        InitializeComponent()
        ' autres initialisations
        myInit()
    End Sub

    'La méthode substituée Dispose du formulaire pour nettoyer la liste des composants.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        ....
    End Sub

    ...

    Private Sub InitializeComponent()
        ....
    End Sub

#End Region

    Private Sub myInit()
        ' init formulaire
        cmbFonctions.SelectedIndex = 0
        btnOuvrir.Enabled = False
        btnFermer.Enabled = True
        btnCalculer.Enabled = False
    End Sub

    Private Sub txtURI_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
txtURI.TextChanged
        ' le contenu du champ de saisie a changé - on fixe l'état du bouton ouvrir
        btnOuvrir.Enabled = txtURI.Text.Trim <> ""
    End Sub

    Private Sub btnOuvrir_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnOuvrir.Click
        ' demande d'ouverture d'une connexion avec le service web
        Try
            ' création d'un objet de type [clientSOAP]
```

```

client = New clientSOAP(txtURI.Text.Trim, False)
' états des boutons
btnOuvrir.Enabled = False
btnFermer.Enabled = True
' l'URI ne peut plus être modifiée
txtURI.ReadOnly = True
' état client
txtErreur.Text = "Liaison au service web ouverte"
Catch ex As Exception
' il y a eu une erreur - on l'affiche
txtErreur.Text = ex.Message
End Try
End Sub

Private Sub btnFermer_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnFermer.Click
' fermeture le la connexion au service web
client.Close()
' états boutons
btnOuvrir.Enabled = True
btnFermer.Enabled = False
' URI
txtURI.ReadOnly = False
' état client
txtErreur.Text = "Liaison au service web fermée"
End Sub

Private Sub btnCalculer_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
btnCalculer.Click
' calcul d'une fonction f(a,b)
' on efface le résultat précédent
txtRésultat.Text = ""
Try
txtRésultat.Text = client.executeFonction(cmbFonctions.Text, txtA.Text.Trim, txtB.Text.Trim)
Catch ex As Exception
' il y a eu une erreur réseau
txtErreur.Text = ex.Message
' on ferme la liaison
btnFermer_Click(Nothing, Nothing)
End Try
End Sub

Private Sub txtA_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
txtA.TextChanged
' changement de la valeur de A
btnCalculer.Enabled = txtA.Text.Trim <> "" And txtB.Text.Trim <> ""
End Sub

Private Sub txtB_TextChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
txtB.TextChanged
' changement de la valeur de B
txtA_TextChanged(Nothing, Nothing)
End Sub

' méthode principale
Public Shared Sub main()
Application.Run(New FormClientSOAP)
End Sub
End Class

```

Là encore la classe *clientSOAP* cache tout l'aspect réseau de l'application. L'application a été construite de la façon suivante :

- l'assemblage *clientSOAP.dll* contenant la classe *clientSOAP* a été placé dans le dossier du projet
- l'interface graphique *clientsoapgui.vb* a été construite avec VS.NET puis compilée dans une fenêtre dos :

```

dos>vbc /r:system.dll /r:system.windows.forms.dll /r:system.drawing.dll /r:clientSOAP.dll clientsoapgui.vb

dos>dir
04/03/2004 09:13          7 168 clientSOAP.dll
04/03/2004 16:44          9 866 clientsoapgui.vb
04/03/2004 16:44         11 264 clientsoapgui.exe

```

L'interface graphique a été ensuite lancée par :

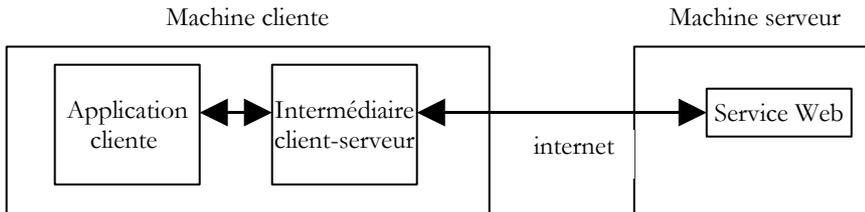
```

dos>clientsoapgui

```

## 9.8 Un client proxy

Rappelons ce qui vient d'être fait. Nous avons créé une classe intermédiaire encapsulant les échanges réseau entre un client et un service web selon le schéma ci-dessous :



La plate-forme .NET pousse cette logique plus loin. Une fois connu le service Web à atteindre, nous pouvons générer automatiquement la classe qui va nous servir d'intermédiaire pour atteindre les fonctions du service Web et qui cachera toute la partie réseau. On appelle cette classe un **proxy** pour le service web pour lequel elle a été générée.

Comment générer la classe proxy d'un service web ? Un service web est toujours accompagné d'un fichier de description au format XML. Si l'URI de notre service web operations est `http://localhost/operations/operations.asmx`, son fichier de description est disponible à l'URL `http://localhost/operations/operations.asmx?wsdl` comme le montre la copie d'écran suivante :

```
Adresse http://localhost/operations/operations.asmx?wsdl
<?xml version="1.0" encoding="utf-8" ?>
- <definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="st.istia.univ-angers.fr"
```

On a là un fichier XML décrivant précisément toutes les fonctions du service web avec pour chacune d'elles le type et le nombre de paramètres, le type du résultat. On appelle ce fichier le fichier WSDL du service parce qu'il utilise le langage **WSDL** (*Web Services Description Language*). A partir de ce fichier, une classe proxy peut être générée à l'aide de l'outil `wsdl` :

```
dos>wsdl http://localhost/operations/operations.asmx?wsdl /language=vb
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 1.1.4322.573]
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Écriture du fichier 'D:\data\devel\vbnet\poly\chap9\clientproxy\operations.vb'.

dos>dir
04/03/2004 17:17                6 663 operations.vb
```

L'outil `wsdl` génère un fichier source VB.NET (option `/language=vb`) portant le nom de la classe implémentant le service web, ici `operations`. Examinons une partie du code généré :

```
'-----
' <autogenerated>
'   This code was generated by a tool.
'   Runtime Version: 1.1.4322.573
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </autogenerated>
'-----

Option Strict Off
Option Explicit On

Imports System
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization

'
'Ce code source a été automatiquement généré par wsdl, Version=1.1.4322.573.
'
```

```
'<remarks/>
<System.Diagnostics.DebuggerStepThroughAttribute(),
  System.ComponentModel.DesignerCategoryAttribute("codé"),
  System.Web.Services.WebServiceBindingAttribute(Name:"operationsSoap", [Namespace]:="st.istia.univ-
angers.fr")>
Public Class operations
  Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

  '<remarks/>
  Public Sub New()
    MyBase.New
    Me.Url = "http://localhost/operations/operations.asmx"
  End Sub

  '<remarks/>
  <System.Web.Services.Protocols.SoapDocumentMethodAttribute("st.istia.univ-angers.fr/ajouter",
RequestNamespace:"st.istia.univ-angers.fr", ResponseNamespace:"st.istia.univ-angers.fr",
Use:=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle:=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>
  Public Function ajouter(ByVal a As Double, ByVal b As Double) As Double
    Dim results() As Object = Me.Invoke("ajouter", New Object() {a, b})
    Return CType(results(0), Double)
  End Function

  '<remarks/>
  Public Function Beginajouter(ByVal a As Double, ByVal b As Double, ByVal callback As
System.AsyncCallback, ByVal asyncState As Object) As System.IAsyncResult
    Return Me.BeginInvoke("ajouter", New Object() {a, b}, callback, asyncState)
  End Function

  '<remarks/>
  Public Function Endajouter(ByVal asyncResult As System.IAsyncResult) As Double
    Dim results() As Object = Me.EndInvoke(asyncResult)
    Return CType(results(0), Double)
  End Function
....
```

Ce code est un peu complexe au premier abord. Nous n'avons pas besoin d'en comprendre les détails pour pouvoir l'utiliser. Examinons tout d'abord la déclaration de la classe :

```
Public Class operations
  Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
```

La classe porte le nom *operations* du service web pour lequel elle a été construite. Elle dérive de la classe *SoapHttpClientProtocol* :

[System.Object](#)  
[System.MarshalByRefObject](#)  
[System.ComponentModel.Component](#)  
[System.Web.Services.Protocols.WebClientProtocol](#)  
[System.Web.Services.Protocols.HttpWebClientProtocol](#)  
**System.Web.Services.Protocols.SoapHttpClientProtocol**

```
Public Class SoapHttpClientProtocol
  Inherits HttpWebClientProtocol
```

Notre classe proxy a un constructeur unique :

```
Public Sub New()
  MyBase.New
  Me.Url = "http://localhost/operations/operations.asmx"
End Sub
```

Le constructeur affecte à l'attribut *url* l'URL du service web associé au proxy. La classe *operations* ci-dessus ne définit pas elle-même l'attribut *url*. Celui-ci est hérité de la classe dont dérive le proxy : *System.Web.Services.Protocols.SoapHttpClientProtocol*. Examinons maintenant ce qui se rapporte à la méthode *ajouter* :

```
Public Function ajouter(ByVal a As Double, ByVal b As Double) As Double
  Dim results() As Object = Me.Invoke("ajouter", New Object() {a, b})
  Return CType(results(0), Double)
End Function
```

On peut constater qu'elle a la même signature que dans le service Web *operations* où elle était définie comme suit :

```
<WebMethod>
Function ajouter(a As Double, b As Double) As Double
    Return a + b
End Function 'ajouter
```

La façon dont cette classe dialogue avec le service Web n'apparaît pas ici. Ce dialogue est entièrement pris en charge par la classe parent *System.Web.Services.Protocols.SoapHttpClientProtocol*. On ne trouve dans le proxy que ce qui le différencie des autres proxy :

- l'URL du service web associé
- la définition des méthodes du service associé.

Pour utiliser les méthodes du service web *operations*, un client n'a besoin que de la classe proxy *operations* générée précédemment. Compilons cette classe dans un fichier *assembly* :

```
dos>vbc /t:library /r:system.web.services.dll /r:system.xml.dll /r:system.dll operations.vb
```

```
dos>dir
04/03/2004 17:17                6 663 operations.vb
04/03/2004 17:24                7 680 operations.dll
```

Maintenant écrivons un client console. Il est appelé sans paramètres et exécute les demandes tapées au clavier :

```
dos>testclientproxy
Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser|toutfaire] a b

ajouter 4 5
résultat=9
soustraire 9 8
résultat=1
multiplier 10 4
résultat=40
diviser 6 7
résultat=0,857142857142857
toutfaire 10 20
résultats=[30,-10,200,0,5]
diviser 5 0
résultat=+Infini
fin
```

Le code du client est le suivant :

```
' espaces de noms
Imports System
Imports System.IO
Imports System.Text.RegularExpressions
Imports System.Collections
Imports Microsoft.VisualBasic

Public Module testClientProxy

' exécute de façon interactive les commandes tapées au clavier
' et les envoie au service web operations
Public Sub Main()
    ' il n'y a plus d'arguments - l'URL du service web étant codée en dur dans le proxy

    ' création d'un dictionnaire des fonctions du service web
    Dim fonctions As String() = {"ajouter", "soustraire", "multiplier", "diviser", "toutfaire"}
    Dim dicoFonctions As New Hashtable
    Dim i As Integer
    For i = 0 To fonctions.Length - 1
        dicoFonctions.Add(fonctions(i), True)
    Next i

    ' on crée un objet proxy operations
    Dim myOperations As operations = Nothing
    Try
        myOperations = New operations
    Catch ex As Exception
        ' erreur de connexion
        erreur("L'erreur suivante s'est produite lors de la connexion au proxy dy service web : " +
ex.Message, 2)
    End Try
```

```

' les demandes de l'utilisateur sont tapées au clavier
' sous la forme fonction a b - elles se terminent avec la commande fin
Dim commande As String = Nothing ' commande tapée au clavier
Dim champs As String() = Nothing ' champs d'une ligne de commande

' invite à l'utilisateur
Console.Out.WriteLine("Tapez vos commandes au format : [ajouter|soustraire|multiplier|diviser|
toutfaire] a b" + ControlChars.Lf)

' qqs données locales
Dim erreurCommande As Boolean
Dim fonction As String
Dim a, b As Double
' boucle de saisie des commandes tapées au clavier
While True
' au départ pas d'erreur
erreurCommande = False
' lecture commande
commande = Console.In.ReadLine().Trim().ToLower()
' fini ?
If commande Is Nothing Or commande = "fin" Then
Exit While
End If
' décomposition de la commande en champs
champs = Regex.Split(commande, "\s+")
Try
' il faut trois champs
If champs.Length <> 3 Then
Throw New Exception
End If
' le champ 0 doit être une fonction reconnue
fonction = champs(0)
If Not dicoFonctions.ContainsKey(fonction) Then
Throw New Exception
End If
' le champ 1 doit être un nombre valide
a = Double.Parse(champs(1))
' le champ 2 doit être un nombre valide
b = Double.Parse(champs(2))
Catch
' commande invalide
Console.Out.WriteLine("syntaxe : [ajouter|soustraire|multiplier|diviser] a b")
erreurCommande = True
End Try
' on fait la demande au service web
If Not erreurCommande Then
Try
Dim résultat As Double
Dim résultats() As Double
If fonction = "ajouter" Then
résultat = myOperations.ajouter(a, b)
Console.Out.WriteLine(("résultat=" + résultat.ToString))
End If
If fonction = "soustraire" Then
résultat = myOperations.soustraire(a, b)
Console.Out.WriteLine(("résultat=" + résultat.ToString))
End If
If fonction = "multiplier" Then
résultat = myOperations.multiplier(a, b)
Console.Out.WriteLine(("résultat=" + résultat.ToString))
End If
If fonction = "diviser" Then
résultat = myOperations.diviser(a, b)
Console.Out.WriteLine(("résultat=" + résultat.ToString))
End If
If fonction = "toutfaire" Then
résultats = myOperations.toutfaire(a, b)
Console.Out.WriteLine(("résultats=[" + résultats(0).ToString + "," + résultats(1).ToString +
", " + -
résultats(2).ToString + "," + résultats(3).ToString + "])")
End If
Catch e As Exception
Console.Out.WriteLine(("L'erreur suivante s'est produite : " + e.Message))
End Try
End If
End While
End Sub

' affichage des erreurs
Public Sub erreur(ByVal msg As String, ByVal exitCode As Integer)
' affichage erreur

```

```

System.Console.Error.WriteLine(msg)
' arrêt avec erreur
Environment.Exit(exitCode)
End Sub
End Module

```

Nous n'examinons que le code propre à l'utilisation de la classe proxy. Tout d'abord un objet proxy *operations* est créé :

```

' on crée un objet proxy operations
Dim myOperations As operations = Nothing
Try
    myOperations = New operations
Catch ex As Exception
    ' erreur de connexion
    erreur("L'erreur suivante s'est produite lors de la connexion au proxy dy service web : " +
ex.Message, 2)
End Try

```

Des lignes *fonction a b* sont tapées au clavier. A partir de ces informations, on appelle les méthodes appropriées du proxy :

```

' on fait la demande au service web
If Not erreurCommande Then
    Try
        Dim résultat As Double
        Dim résultats() As Double
        If fonction = "ajouter" Then
            résultat = myOperations.ajouter(a, b)
            Console.Out.WriteLine(("résultat=" + résultat.ToString))
        End If
        If fonction = "soustraire" Then
            résultat = myOperations.soustraire(a, b)
            Console.Out.WriteLine(("résultat=" + résultat.ToString))
        End If
        If fonction = "multiplier" Then
            résultat = myOperations.multiplier(a, b)
            Console.Out.WriteLine(("résultat=" + résultat.ToString))
        End If
        If fonction = "diviser" Then
            résultat = myOperations.diviser(a, b)
            Console.Out.WriteLine(("résultat=" + résultat.ToString))
        End If
        If fonction = "toutfaire" Then
            résultats = myOperations.toutfaire(a, b)
            Console.Out.WriteLine(("résultats=[" + résultats(0).ToString + "," + résultats(1).ToString +
", " + -
            résultats(2).ToString + "," + résultats(3).ToString + "]"))
        End If
    Catch e As Exception
        Console.Out.WriteLine(("L'erreur suivante s'est produite : " + e.Message))
    End Try

```

On traite ici pour la première fois, l'opération *toutfaire* qui fait les quatre opérations. Elle avait été ignorée jusqu'à maintenant car elle envoie un tableau de nombres encapsulé dans une enveloppe XML plus compliquée à gérer que les réponses XML simples des autres fonctions ne délivrant qu'un unique résultat. On voit qu'ici avec la classe proxy, utiliser la méthode *toutfaire* n'est pas plus compliqué qu'utiliser les autres méthodes. L'application a été compilée dans une fenêtre dos de la façon suivante :

```

dos>vbc /r:operations.dll /r:system.dll /r:system.web.services.dll testClientProxy.vb

dos>dir
04/03/2004 17:17          6 663 operations.vb
04/03/2004 17:24          7 680 operations.dll
04/03/2004 17:41          4 099 testClientProxy.vb
04/03/2004 17:41          5 632 testClientProxy.exe

```

## 9.9 Configurer un service Web

Un service Web peut avoir besoin d'informations de configuration pour s'initialiser correctement. Avec le serveur IIS, ces informations peuvent être placées dans un fichier appelé *web.config* et situé dans le même dossier que le service web. Supposons qu'on veuille créer un service web ayant besoin de deux informations pour s'initialiser : un nom et un âge. Ces deux informations peuvent être placées dans le fichier *web.config* sous la forme suivante :

```

<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
  <appSettings>

```

```

    <add key="nom" value="tintin"/>
    <add key="age" value="27"/>
  </appSettings>
</configuration>

```

Les paramètres d'initialisation sont placées dans une enveloppe XML :

```

<configuration>
  <appSettings>
  ...
  </appSettings>
</configuration>

```

Un paramètre d'initialisation de nom *P* ayant la valeur *V* sera déclarée avec la ligne :

```
<add key="P" value="V"/>
```

Comment le service Web récupère-t-il ces informations ? Lorsque IIS charge un service web, il regarde s'il y a dans le même dossier un fichier *web.config*. Si oui, il le lit. La valeur *V* d'un paramètre *P* est obtenue par l'instruction :

```
String P=ConfigurationSettings.AppSettings["V"];
```

où *ConfigurationSettings* est une classe dans l'espace de noms *System.Configuration*.

Vérifions cette technique sur le service web suivant :

```

<%@ WebService language="VB" class=personne %>

Imports System.Web.Services
imports System.Configuration

<WebService([Namespace] := "st.istia.univ-angers.fr")> _
Public Class personne
  Inherits WebService

  ' attributs
  Private nom As String
  Private age As Integer

  ' constructeur
  Public Sub New()
    ' init attributs
    nom = ConfigurationSettings.AppSettings("nom")
    age = Integer.Parse(ConfigurationSettings.AppSettings("age"))
  End Sub

  <WebMethod>
  Function id() As String
    Return "[" + nom + "," + age.ToString + "]"
  End Function

End Class

```

Le service web *personne* a deux attributs *nom* et *age* qui sont initialisés dans son constructeur sans paramètres à partir des valeurs lues dans le fichier de configuration *web.config* du service *personne*. Ce fichier est le suivant :

```

<configuration>
  <appSettings>
    <add key="nom" value="tintin"/>
    <add key="age" value="27"/>
  </appSettings>
</configuration>

```

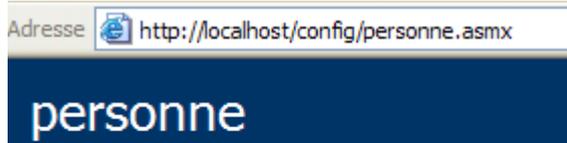
Le service web a par ailleurs une **<WebMethod> id** sans paramètres et qui se contente de rendre les attributs *nom* et *age*. Le service est enregistré dans le fichier source *personne.asmx* qui est placé avec son fichier de configuration dans le dossier *c:\inetpub\wwwroot\st\personne* :

```

dos>dir
09/03/2004  08:25                632 personne.asmx
09/03/2004  08:08                186 web.config

```

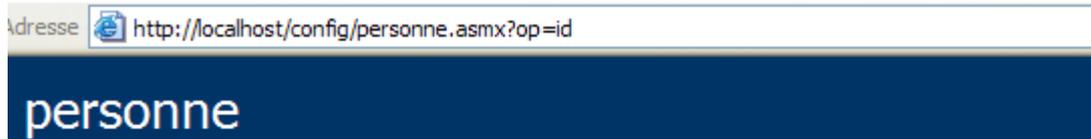
Associons un dossier virtuel IIS /config au dossier physique précédent. Lançons IIS puis avec un navigateur demandons l'url *http://localhost/config/personne.asmx* du service *personne* :



Les opérations suivantes sont prises en charge

- [id](#)

Suivons le lien de l'unique méthode **id** :



Cliquez [ici](#) pour une liste complète des opérations.

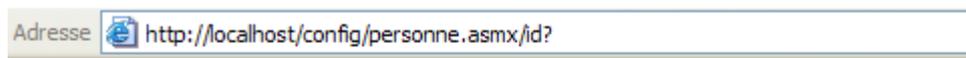
## id

### Test

Pour tester l'opération en utilisant le protocole HTTP GET, cliquez sur le bouton 'Appeler'.



La méthode *id* n'a pas de paramètres. Utilisons le bouton *Appeler* :



```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="st.istia.univ-angers.fr">[tintin,27]</string>
```

Nous avons bien récupéré les informations placées dans le fichier *web.config* du service.

## 9.10 Le service Web IMPOTS

Nous reprenons l'application IMPOTS maintenant bien connue. La dernière fois que nous avons travaillé avec, nous en avons fait un serveur distant qu'on pouvait appeler sur l'internet. Nous en faisons maintenant un service web.

### 9.10.1 Le service web

Nous partons de la classe *impôt* créée dans le chapitre sur les bases de données et qui se construit à partir des informations contenues dans une base de données ODBC :

```
' options
Option Strict On
Option Explicit On

' espaces de noms
Imports System
Imports System.Data
Imports Microsoft.Data.Odbc
Imports System.Collections

Public Class impôt
' les données nécessaires au calcul de l'impôt
' proviennent d'une source extérieure
Private limites(), coeffR(), coeffN() As Decimal
```

```

' constructeur
Public Sub New(ByVal LIMITES() As Decimal, ByVal COEFFR() As Decimal, ByVal COEFFN() As Decimal)
' on vérifie que les 3 tableaux ont la même taille
Dim OK As Boolean = LIMITES.Length = COEFFR.Length And LIMITES.Length = COEFFN.Length
If Not OK Then
Throw New Exception("Les 3 tableaux fournis n'ont pas la même taille(" & LIMITES.Length & "," &
COEFFR.Length & "," & COEFFN.Length & ")")
End If
' c'est bon
Me.limiteS = LIMITES
Me.coeffR = COEFFR
Me.coeffN = COEFFN
End Sub

' constructeur 2
Public Sub New(ByVal DSNimpots As String, ByVal Timpots As String, ByVal colLimiteS As String, ByVal
colCoeffR As String, ByVal colCoeffN As String)
' initialise les trois tableaux limiteS, coeffR, coeffN à partir
' du contenu de la table Timpots de la base ODBC DSNimpots
' colLimiteS, colCoeffR, colCoeffN sont les trois colonnes de cette table
' peut lancer une exception
Dim connectString As String = "DSN=" + DSNimpots + ";" ' chaîne de connexion à la base
Dim impotsConn As OdbcConnection = Nothing ' la connexion
Dim sqlCommand As OdbcCommand = Nothing ' la commande SQL
' la requête SELECT
Dim selectCommand As String = "select " + colLimiteS + "," + colCoeffR + "," + colCoeffN + " from " +
Timpots
' tableaux pour récupérer les données
Dim tLimiteS As New ArrayList
Dim tCoeffR As New ArrayList
Dim tCoeffN As New ArrayList

' on tente d'accéder à la base de données
impotsConn = New OdbcConnection(connectString)
impotsConn.Open()
' on crée un objet command
sqlCommand = New OdbcCommand(selectCommand, impotsConn)
' on exécute la requête
Dim myReader As OdbcDataReader = sqlCommand.ExecuteReader()
' Exploitation de la table récupérée
While myReader.Read()
' les données de la ligne courante sont mis dans les tableaux
tLimiteS.Add(myReader(colLimiteS))
tCoeffR.Add(myReader(colCoeffR))
tCoeffN.Add(myReader(colCoeffN))
End While
' libération des ressources
myReader.Close()
impotsConn.Close()

' les tableaux dynamiques sont mis dans des tableaux statiques
Me.limiteS = New Decimal(tLimiteS.Count) {}
Me.coeffR = New Decimal(tCoeffR.Count) {}
Me.coeffN = New Decimal(tCoeffN.Count) {}
Dim i As Integer
For i = 0 To tLimiteS.Count - 1
limiteS(i) = Decimal.Parse(tLimiteS(i).ToString())
coeffR(i) = Decimal.Parse(tCoeffR(i).ToString())
coeffN(i) = Decimal.Parse(tCoeffN(i).ToString())
Next i
End Sub

' calcul de l'impôt
Public Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Integer)
As Long
' calcul du nombre de parts
Dim nbParts As Decimal
If marié Then
nbParts = CDec(nbEnfants) / 2 + 2
Else
nbParts = CDec(nbEnfants) / 2 + 1
End If
If nbEnfants >= 3 Then
nbParts += 0.5D
End If
' calcul revenu imposable & Quotient familial
Dim revenu As Decimal = 0.72D * salaire
Dim QF As Decimal = revenu / nbParts
' calcul de l'impôt
limiteS((limiteS.Length - 1)) = QF + 1
Dim i As Integer = 0

```

```

While QF > limites(i)
    i += 1
End While
' retour résultat
Return CLng(revenu * coeffR(i) - nbParts * coeffN(i))
End Function
End Class

```

Dans le service web, on ne peut utiliser qu'un constructeur sans paramètres. Aussi le constructeur de la classe va-t-il devenir le suivant :

```

' constructeur
Public Sub New()
    ' initialise les trois tableaux limites, coeffR, coeffN à partir
    ' du contenu de la table Timpots de la base ODBC DSNimpots
    ' colLimites, colCoeffR, colCoeffN sont les trois colonnes de cette table
    ' peut lancer une exception

    ' on récupère les paramètres de configuration du service
    Dim DSNimpots As String = ConfigurationSettings.AppSettings("DSN")
    Dim Timpots As String = ConfigurationSettings.AppSettings("TABLE")
    Dim colLimites As String = ConfigurationSettings.AppSettings("COL_LIMITES")
    Dim colCoeffR As String = ConfigurationSettings.AppSettings("COL_COEFFR")
    Dim colCoeffN As String = ConfigurationSettings.AppSettings("COL_COEFFN")

    ' on exploite la base de données
    Dim connectString As String = "DSN=" + DSNimpots + ";" ' chaîne de connexion à la base

```

Les cinq paramètres du constructeur de la classe précédente sont maintenant lus dans le fichier *web.config* du service. Le code du fichier source *impots.asmx* est le suivant. Il reprend la majeure partie du code précédent. Nous nous sommes contentés d'encadrer les portions de code propres au service web :

```

<%@ WebService language="VB" class=impots %>

' création d'un servie web impots
Imports System
Imports System.Data
Imports Microsoft.Data.Odbc
Imports System.Collections
Imports System.Configuration
Imports System.Web.Services

<WebService([Namespace]:= "st.istia.univ-angers.fr")> _
Public Class impôt
    Inherits WebService

    ' les données nécessaires au calcul de l'impôt
    ' proviennent d'une source extérieure
    Private limites(), coeffR(), coeffN() As Decimal
    Private OK As Boolean = False
    Private errMessage As String = ""

    ' constructeur
    Public Sub New()
        ' initialise les trois tableaux limites, coeffR, coeffN à partir
        ' du contenu de la table Timpots de la base ODBC DSNimpots
        ' colLimites, colCoeffR, colCoeffN sont les trois colonnes de cette table
        ' peut lancer une exception

        ' on récupère les paramètres de configuration du service
        Dim DSNimpots As String = ConfigurationSettings.AppSettings("DSN")
        Dim Timpots As String = ConfigurationSettings.AppSettings("TABLE")
        Dim colLimites As String = ConfigurationSettings.AppSettings("COL_LIMITES")
        Dim colCoeffR As String = ConfigurationSettings.AppSettings("COL_COEFFR")
        Dim colCoeffN As String = ConfigurationSettings.AppSettings("COL_COEFFN")

        ' on exploite la base de données
        Dim connectString As String = "DSN=" + DSNimpots + ";" ' chaîne de connexion à la base
        Dim impotsConn As OdbcConnection = Nothing ' la connexion
        Dim sqlCommand As OdbcCommand = Nothing ' la commande SQL
        Dim myReader As OdbcDataReader ' lecteur de données Odbc

        ' la requête SELECT
        Dim selectCommand As String = "select " + colLimites + "," + colCoeffR + "," + colCoeffN + " from " +
Timpots

        ' tableaux pour récupérer les données
        Dim tLimites As New ArrayList

```

```

Dim tCoeffR As New ArrayList
Dim tCoeffN As New ArrayList

' on tente d'accéder à la base de données
Try
    impotsConn = New OdbcConnection(connectString)
    impotsConn.Open()
    ' on crée un objet command
    sqlCommand = New OdbcCommand(selectCommand, impotsConn)
    ' on exécute la requête
    myReader = sqlCommand.ExecuteReader()
    ' Exploitation de la table récupérée
    While myReader.Read()
        ' les données de la ligne courante sont mis dans les tableaux
        tLimites.Add(myReader(colLimites))
        tCoeffR.Add(myReader(colCoeffR))
        tCoeffN.Add(myReader(colCoeffN))
    End While
    ' libération des ressources
    myReader.Close()
    impotsConn.Close()

    ' les tableaux dynamiques sont mis dans des tableaux statiques
    Me.limites = New Decimal(tLimites.Count) {}
    Me.coeffR = New Decimal(tCoeffR.Count) {}
    Me.coeffN = New Decimal(tCoeffN.Count) {}
    Dim i As Integer
    For i = 0 To tLimites.Count - 1
        limites(i) = Decimal.Parse(tLimites(i).ToString())
        coeffR(i) = Decimal.Parse(tCoeffR(i).ToString())
        coeffN(i) = Decimal.Parse(tCoeffN(i).ToString())
    Next i
    ' c'est bon
    OK = True
    errMessage = ""
Catch ex As Exception
    ' erreur
    OK = False
    errMessage += "[" + ex.Message + "]"
End Try
End Sub

' calcul de l'impôt
<WebMethod()>
Function calculer(ByVal marié As Boolean, ByVal nbEnfants As Integer, ByVal salaire As Integer) As Long
    ' calcul du nombre de parts
    Dim nbParts As Decimal
    If marié Then
        nbParts = CDec(nbEnfants) / 2 + 2
    Else
        nbParts = CDec(nbEnfants) / 2 + 1
    End If
    If nbEnfants >= 3 Then
        nbParts += 0.5D
    End If
    ' calcul revenu imposable & Quotient familial
    Dim revenu As Decimal = 0.72D * salaire
    Dim QF As Decimal = revenu / nbParts
    ' calcul de l'impôt
    limites((limites.Length - 1)) = QF + 1
    Dim i As Integer = 0
    While QF > limites(i)
        i += 1
    End While
    ' retour résultat
    Return CLng(revenu * coeffR(i) - nbParts * coeffN(i))
End Function

' id
<WebMethod()>
Function id() As String
    ' pour voir si tout est OK
    Return "[" + OK + "," + errMessage + "]"
End Function
End Class

```

Expliquons les quelques modifications faites à la classe *impots* en-dehors de celles nécessaires pour en faire un service web :

- la lecture de la base de données dans le constructeur peut échouer. Aussi avons-nous ajouté deux attributs à notre classe et une méthode :
  - le booléen *OK* est à *vrai* si la base a pu être lue, à *faux* sinon
  - la chaîne *errorMessage* contient un message d'erreur si la base de données n'a pu être lue.
  - la méthode *id* sans paramètres permet d'obtenir la valeur ces deux attributs.
- pour gérer l'erreur éventuelle d'accès à la base de données, la partie du code du constructeur concernée par cet accès a été entourée d'un *try-catch*.

Le fichier *web.config* de configuration du service est le suivant :

```
<configuration>
  <appSettings>
    <add key="DSN" value="mysql-impots" />
    <add key="TABLE" value="timpots" />
    <add key="COL_LIMITES" value="limites" />
    <add key="COL_COEFFR" value="coeffr" />
    <add key="COL_COEFFN" value="coeffn" />
  </appSettings>
</configuration>
```

Lors d'un premier essai de chargement du service *impots*, le compilateur a déclaré qu'il ne trouvait pas l'espace de noms *Microsoft.Data.Odbc* utilisé dans la directive :

```
Imports Microsoft.Data.Odbc
```

Après consultation de la documentation

- une directive de compilation a été ajoutée dans *web.config* pour indiquer qu'il fallait utiliser l'assembly *Microsoft.Data.odbc*
- une copie du fichier *microsoft.data.odbc.dll* a été placée dans le dossier *bin* du projet. Celui-ci est systématiquement exploré par le compilateur d'un service web lorsqu'il recherche un "assembly".

D'autres solutions semblent possibles mais n'ont pas été creusées ici. Le fichier de configuration est donc devenu :

```
<configuration>
  <appSettings>
    <add key="DSN" value="mysql-impots" />
    <add key="TABLE" value="timpots" />
    <add key="COL_LIMITES" value="limites" />
    <add key="COL_COEFFR" value="coeffr" />
    <add key="COL_COEFFN" value="coeffn" />
  </appSettings>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="Microsoft.Data.Odbc" />
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

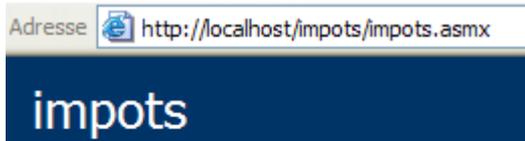
Le contenu du dossier *impots\bin* :

```
dos>dir impots\bin
30/01/2002  02:02                327 680 Microsoft.Data.Odbc.dll
```

Le service et son fichier de configuration ont été placés dans *impots* :

```
dos>dir impots
09/03/2004  10:13                4 669 impots.asmx
09/03/2004  10:19                431 web.config
```

Le dossier physique du service web a été associé au dossier virtuel */impots* de IIS. La page du service est alors la suivante :



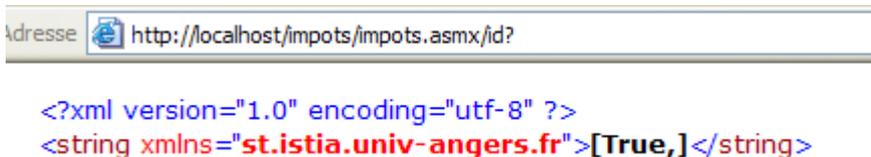
Les opérations suivantes sont prises en charge :

- [calculer](#)
- [id](#)

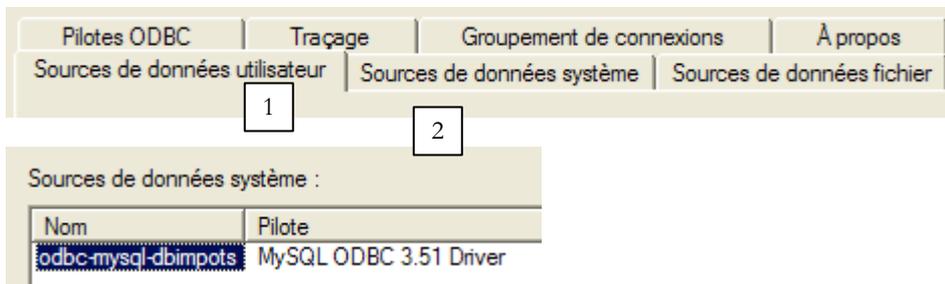
Si on suit le lien *id* :



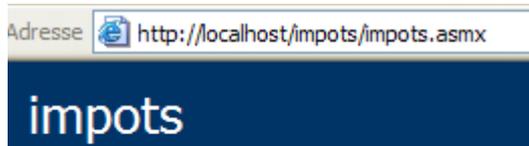
Si on utilise le bouton *Appeler* :



Le résultat précédent affiche les valeurs des attributs *OK* (true) et *errorMessage* (""). Dans cet exemple, la base a été chargée correctement. Ça n'a pas toujours été le cas et c'est pourquoi nous avons ajouté la méthode *id* pour avoir accès au message d'erreur. L'erreur était que le nom DSN de la base avait été définie comme **DSN utilisateur** alors qu'il fallait le définir comme **DSN système**. Cette distinction se fait dans le gestionnaire de sources ODBC 32 bits :



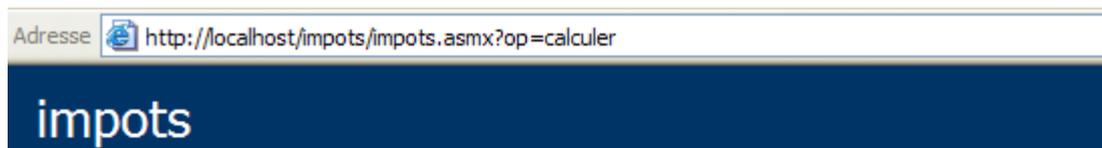
Revenons à la page du service :



Les opérations suivantes sont prises en charge :

- [calculer](#)
- [id](#)

Suivons le lien *calculer* :



Cliquez [ici](#) pour une liste complète des opérations.

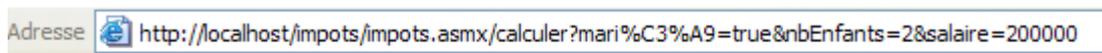
## calculer

### Test

Pour tester l'opération en utilisant le protocole HTTP GET, cliquez sur le bouton 'Appeler'.

Paramètre	Valeur
marié:	<input type="text" value="true"/>
nbEnfants:	<input type="text" value="2"/>
salaire:	<input type="text" value="200000"/>

Nous définissons les paramètres de l'appel et nous exécutons celui-ci :



```
<?xml version="1.0" encoding="utf-8" ?>
<long xmlns="st.istia.univ-angers.fr">22504</long>
```

Le résultat est correct.

## 9.10.2 Générer le proxy du service impots

Maintenant que nous avons un service web *impots* opérationnel, nous pouvons générer sa classe proxy. On rappelle que celle-ci sera utilisée par des applications clientes pour atteindre le service web *impots* de façon transparente. On utilise d'abord l'utilitaire *wSDL* pour générer le fichier source de la classe proxy puis celui-ci est compilé dans un une dll.

```
dos>wsdl /language=vb http://localhost/impots/impots.asmx
Microsoft (R) Web Services Description Language Utility
[Microsoft (R) .NET Framework, Version 1.1.4322.573]
Copyright (C) Microsoft Corporation 1998-2002. All rights reserved.

Écriture du fichier 'D:\data\serge\devel\vbnet\poly\chap9\impots\impots.vb'.

D:\data\serge\devel\vbnet\poly\chap9\impots>dir
09/03/2004 10:20 <REP> bin
09/03/2004 10:58 4 651 impots.asmx
09/03/2004 11:05 3 364 impots.vb
09/03/2004 10:19 431 web.config
```

```

dos>vbc /t:library /r:system.dll /r:system.web.services.dll /r:system.xml.dll impots.vb
Compilateur Microsoft (R) Visual Basic .NET version 7.10.3052.4
pour Microsoft (R) .NET Framework version 1.1.4322.573
Copyright (C) Microsoft Corporation 1987-2002. Tous droits réservés.

```

```

dos>dir
09/03/2004 10:20 <REP> bin
09/03/2004 10:58 4 651 impots.asmx
09/03/2004 11:09 5 120 impots.dll
09/03/2004 11:05 3 364 impots.vb
09/03/2004 10:19 431 web.config

```

### 9.10.3 Utiliser le proxy avec un client

Dans le chapitre sur les bases de données nous avons créé une application console permettant le calcul de l'impôt :

```

dos>dir
27/02/2004 16:56 5 120 impots.dll
27/02/2004 17:12 3 586 impots.vb
27/02/2004 17:08 6 144 testimpots.exe
27/02/2004 17:18 3 328 testimpots.vb

dos>testimpots
pg DSNimpots tabImpots colLimites colCoeffR colCoeffN

dos>testimpots odbc-mysql-dbimpots impots limites coeffr coeffn
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 200000
impôt=22504 F

```

Le programme *testimpots* utilisait alors la classe *impôt* classique celle contenue dans le fichier *impots.dll*. Le code du programme *testimpots.vb* était le suivant :

```

Option Explicit On
Option Strict On

' espaces de noms
Imports System
Imports Microsoft.VisualBasic

' pg de test
Module testimpots
Sub Main(ByVal arguments() As String)
' programme interactif de calcul d'impôt
' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
' le programme affiche alors l'impôt à payer
Const syntaxe1 As String = "pg DSNimpots tabImpots colLimites colCoeffR colCoeffN"
Const syntaxe2 As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour
marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire
: salaire annuel en F"

' vérification des paramètres du programme
If arguments.Length <> 5 Then
' msg d'erreur
Console.Error.WriteLine(syntaxe1)
' fin
Environment.Exit(1)
End If 'if
' on récupère les arguments
Dim DSNimpots As String = arguments(0)
Dim tabImpots As String = arguments(1)
Dim colLimites As String = arguments(2)
Dim colCoeffR As String = arguments(3)
Dim colCoeffN As String = arguments(4)

' création d'un objet impôt
Dim objImpôt As impôt = Nothing
Try
objImpôt = New impôt(DSNimpots, tabImpots, colLimites, colCoeffR, colCoeffN)
Catch ex As Exception
Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
Environment.Exit(2)
End Try

' boucle infinie
While True
' au départ pas d'erreurs
Dim erreur As Boolean = False

```

```

' on demande les paramètres du calcul de l'impôt
Console.Out.WriteLine("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour
arrêter :")
Dim paramètres As String = Console.In.ReadLine().Trim()

' qq chose à faire ?
If paramètres Is Nothing Or paramètres = "" Then
    Exit While
End If

' vérification du nombre d'arguments dans la ligne saisie
Dim args As String() = paramètres.Split(Nothing)
Dim nbParamètres As Integer = args.Length
If nbParamètres <> 3 Then
    Console.Error.WriteLine(syntaxe2)
    erreur = True
End If
Dim marié As String
Dim nbEnfants As Integer
Dim salaire As Integer
If Not erreur Then
    ' vérification de la validité des paramètres
    ' marié
    marié = args(0).ToLower()
    If marié <> "o" And marié <> "n" Then
        Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument marié incorrect : tapez o ou
n"))
        erreur = True
    End If
    ' nbEnfants
    nbEnfants = 0
    Try
        nbEnfants = Integer.Parse(args(1))
        If nbEnfants < 0 Then
            Throw New Exception
        End If
    Catch
        Console.Error.WriteLine(syntaxe2 + "\nArgument nbEnfants incorrect : tapez un entier positif ou
nul")
        erreur = True
    End Try
    ' salaire
    salaire = 0
    Try
        salaire = Integer.Parse(args(2))
        If salaire < 0 Then
            Throw New Exception
        End If
    Catch
        Console.Error.WriteLine(syntaxe2 + "\nArgument salaire incorrect : tapez un entier positif ou
nul")
        erreur = True
    End Try
End If
If Not erreur Then
    ' les paramètres sont corrects - on calcule l'impôt
    Console.Out.WriteLine(("impôt=" & objImpôt.calculer(marié = "o", nbEnfants, salaire).ToString + "
F"))
End If
End While
End Sub
End Module

```

Nous reprenons le même programme pour lui faire utiliser maintenant le service web *impots* au travers de la classe proxy *impots* créée précédemment. Nous sommes obligés de modifier quelque peu le code :

- alors que la classe *impôt* d'origine avait un constructeur à cinq arguments, la classe proxy *impots* a un constructeur sans paramètres. Les cinq paramètres, nous l'avons vu, sont maintenant fixés dans le fichier de configuration du service web.
- il n'y a donc plus besoin de passer ces cinq paramètres en arguments au programme test

Le nouveau code est le suivant :

```

Imports System
Imports Microsoft.VisualBasic

' pg de test
Module testimpots

    Public Sub Main(ByVal arguments() As String)
        ' programme interactif de calcul d'impôt
    End Sub
End Module

```

```

' l'utilisateur tape trois données au clavier : marié nbEnfants salaire
' le programme affiche alors l'impôt à payer
Const syntaxe2 As String = "syntaxe : marié nbEnfants salaire" + ControlChars.Lf + "marié : o pour
marié, n pour non marié" + ControlChars.Lf + "nbEnfants : nombre d'enfants" + ControlChars.Lf + "salaire
: salaire annuel en F"

' création d'un objet impôt
Dim objImpôt As impôt = Nothing
Try
    objImpôt = New impôt
Catch ex As Exception
    Console.Error.WriteLine(("L'erreur suivante s'est produite : " + ex.Message))
    Environment.Exit(2)
End Try

' boucle infinie
Dim erreur As Boolean
While True
    ' au départ pas d'erreur
    erreur = False
    ' on demande les paramètres du calcul de l'impôt
    Console.Out.Write("Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour
arrêter :")
    Dim paramètres As String = Console.In.ReadLine().Trim()
    ' qq chose à faire ?
    If paramètres Is Nothing Or paramètres = "" Then
        Exit While
    End If
    ' vérification du nombre d'arguments dans la ligne saisie
    Dim args As String() = paramètres.Split(Nothing)
    Dim nbParamètres As Integer = args.Length
    If nbParamètres <> 3 Then
        Console.Error.WriteLine(syntaxe2)
        erreur = True
    End If
    If Not erreur Then
        ' vérification de la validité des paramètres
        ' marié
        Dim marié As String = args(0).ToLower()
        If marié <> "o" And marié <> "n" Then
            Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument marié incorrect : tapez o ou
n"))
            erreur = True
        End If
        ' nbEnfants
        Dim nbEnfants As Integer = 0
        Try
            nbEnfants = Integer.Parse(args(1))
            If nbEnfants < 0 Then
                Throw New Exception
            End If
        Catch
            Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument nbEnfants incorrect : tapez un
entier positif ou nul"))
            erreur = True
        End Try
        ' salaire
        Dim salaire As Integer = 0
        Try
            salaire = Integer.Parse(args(2))
            If salaire < 0 Then
                Throw New Exception
            End If
        Catch
            Console.Error.WriteLine((syntaxe2 + ControlChars.Lf + "Argument salaire incorrect : tapez un
entier positif ou nul"))
            erreur = True
        End Try
        ' si les paramètres sont corrects - on calcule l'impôt
        If Not erreur Then Console.Out.WriteLine(("impôt=" + objImpôt.calculer(marié = "o", nbEnfants,
salaire).ToString + " F"))
        End If
    End While
End Sub
End Module

```

Nous avons le proxy *impots.dll* et le source *testimpots* dans le même dossier.

```

dos>dir
09/03/2004 11:28 <REP> bin
09/03/2004 11:09 5 120 impots.dll

```

```
09/03/2004 11:34      3 396 testimpots.vb
09/03/2004 10:19      431 web.config
```

Nous compilons le source *testimpots.vb* :

```
dos>vbc /r:impots.dll /r:microsoft.visualbasic.dll /r:system.web.services.dll /r:system.dll testimpots.vb
```

```
dos>dir
09/03/2004 11:28      <REP>          bin
09/03/2004 11:09          5 120 impots.dll
09/03/2004 11:05          3 364 impots.vb
09/03/2004 11:35          5 632 testimpots.exe
09/03/2004 11:34          3 396 testimpots.vb
09/03/2004 10:19          431 web.config
```

puis l'exécutons :

```
dos>testimpots
Paramètres du calcul de l'impôt au format marié nbEnfants salaire ou rien pour arrêter :o 2 200000
impôt=22504 F
```

Nous obtenons bien le résultat attendu.

# 10. A suivre...

Il resterait des thèmes importants à couvrir. En voici trois :

1. les objets **DataSet** qui permettent de gérer une base de données en mémoire, de l'exporter ou l'importer au format XML
2. une étude de XML avec les classes .NET permettant de gérer les documents XML
3. la programmation Web avec les pages et contrôles ASP.NET

A lui seul, le point 3 mérite un polycopié. Les points 1 et 2 devraient être ajoutés progressivement à ce présent document.

## **1. LES BASES DU LANGAGE VB.NET.....7**

<b>1.1</b>	<b>INTRODUCTION.....</b>	<b>7</b>
<b>1.2</b>	<b>LES DONNÉES DE VB.NET.....</b>	<b>7</b>
1.2.1	LES TYPES DE DONNÉES PRÉDÉFINIS.....	7
1.2.2	NOTATION DES DONNÉES LITTÉRALES.....	8
1.2.3	DÉCLARATION DES DONNÉES.....	8
1.2.3.1	Rôle des déclarations.....	8
1.2.3.2	Déclaration des constantes.....	9
1.2.3.3	Déclaration des variables.....	9
1.2.4	LES CONVERSIONS ENTRE NOMBRES ET CHAÎNES DE CARACTÈRES.....	9
1.2.5	LES TABLEAUX DE DONNÉES.....	10
<b>1.3</b>	<b>LES INSTRUCTIONS ÉLÉMENTAIRES DE VB.NET.....</b>	<b>13</b>
1.3.1	ÉCRITURE SUR ÉCRAN.....	13
1.3.2	LECTURE DE DONNÉES TAPÉES AU CLAVIER.....	14
1.3.3	EXEMPLE D'ENTRÉES-SORTIES.....	14
1.3.4	REDIRECTION DES E/S.....	14
1.3.5	AFFECTATION DE LA VALEUR D'UNE EXPRESSION À UNE VARIABLE.....	16
1.3.5.1	Liste des opérateurs.....	16
1.3.5.2	Expression arithmétique.....	16
1.3.5.3	Priorités dans l'évaluation des expressions arithmétiques.....	17
1.3.5.4	Expressions relationnelles.....	17
1.3.5.5	Expressions booléennes.....	18
1.3.5.6	Traitement de bits.....	18
1.3.5.7	Opérateur associé à une affectation.....	19
1.3.5.8	Priorité générale des opérateurs.....	19
1.3.5.9	Les conversions de type.....	19
<b>1.4</b>	<b>LES INSTRUCTIONS DE CONTRÔLE DU DÉROULEMENT DU PROGRAMME.....</b>	<b>21</b>
1.4.1	ARRÊT.....	21
1.4.2	STRUCTURE DE CHOIX SIMPLE.....	21
1.4.3	STRUCTURE DE CAS.....	22
1.4.4	STRUCTURE DE RÉPÉTITION.....	23
1.4.4.1	Nombre de répétitions connu.....	23
1.4.4.2	Nombre de répétitions inconnu.....	24
1.4.4.3	Instructions de gestion de boucle.....	24
<b>1.5</b>	<b>LA STRUCTURE D'UN PROGRAMME VB.NET.....</b>	<b>25</b>
<b>1.6</b>	<b>COMPILATION ET EXÉCUTION D'UN PROGRAMME VB.NET.....</b>	<b>26</b>
<b>1.7</b>	<b>L'EXEMPLE IMPOTS.....</b>	<b>27</b>
<b>1.8</b>	<b>ARGUMENTS DU PROGRAMME PRINCIPAL.....</b>	<b>29</b>
<b>1.9</b>	<b>LES ÉNUMÉRATIONS.....</b>	<b>30</b>
<b>1.10</b>	<b>LA GESTION DES EXCEPTIONS.....</b>	<b>31</b>
<b>1.11</b>	<b>PASSAGE DE PARAMÈTRES À UNE FONCTION.....</b>	<b>33</b>
1.11.1	PASSAGE PAR VALEUR.....	33
1.11.2	PASSAGE PAR RÉFÉRENCE.....	34

## **2. CLASSES, STRUCTURES, INTERFACES.....35**

<b>2.1</b>	<b>L' OBJET PAR L'EXEMPLE.....</b>	<b>35</b>
2.1.1	GÉNÉRALITÉS.....	35
2.1.2	DÉFINITION DE LA CLASSE PERSONNE.....	35
2.1.3	LA MÉTHODE INITIALISE.....	36
2.1.4	L'OPÉRATEUR NEW.....	36
2.1.5	LE MOT CLÉ ME.....	37
2.1.6	UN PROGRAMME DE TEST.....	37
2.1.7	UTILISER UN FICHIER DE CLASSES COMPILÉES (ASSEMBLY).....	38
2.1.8	UNE AUTRE MÉTHODE INITIALISE.....	39
2.1.9	CONSTRUCTEURS DE LA CLASSE PERSONNE.....	39
2.1.10	LES RÉFÉRENCES D'OBJETS.....	41
2.1.11	LES OBJETS TEMPORAIRES.....	42
2.1.12	MÉTHODES DE LECTURE ET D'ÉCRITURE DES ATTRIBUTS PRIVÉS.....	42

<a href="#">2.1.13</a>	LES PROPRIÉTÉS.....	43
<a href="#">2.1.14</a>	LES MÉTHODES ET ATTRIBUTS DE CLASSE.....	45
<a href="#">2.1.15</a>	PASSAGE D'UN OBJET À UNE FONCTION.....	47
<a href="#">2.1.16</a>	UN TABLEAU DE PERSONNES.....	48
<b><a href="#">2.2</a></b>	<b>L'HÉRITAGE PAR L'EXEMPLE.....</b>	<b>48</b>
<a href="#">2.2.1</a>	GÉNÉRALITÉS.....	48
<a href="#">2.2.2</a>	CONSTRUCTION D'UN OBJET ENSEIGNANT.....	50
<a href="#">2.2.3</a>	SURCHARGE D'UNE MÉTHODE OU D'UNE PROPRIÉTÉ.....	52
<a href="#">2.2.4</a>	LE POLYMORPHISME.....	53
<a href="#">2.2.5</a>	REDÉFINITION ET POLYMORPHISME.....	53
<b><a href="#">2.3</a></b>	<b>DÉFINIR UN INDEXEUR POUR UNE CLASSE.....</b>	<b>56</b>
<b><a href="#">2.4</a></b>	<b>LES STRUCTURES.....</b>	<b>61</b>
<b><a href="#">2.5</a></b>	<b>LES INTERFACES.....</b>	<b>64</b>
<b><a href="#">2.6</a></b>	<b>LES ESPACES DE NOMS.....</b>	<b>68</b>
<b><a href="#">2.7</a></b>	<b>L'EXEMPLE IMPOTS.....</b>	<b>70</b>

### **[3. CLASSES .NET D'USAGE COURANT.....](#)** **74**

<b><a href="#">3.1</a></b>	<b>CHECHER DE L'AIDE AVEC SDK.NET.....</b>	<b>74</b>
<a href="#">3.1.1</a>	WINCV.....	74
<b><a href="#">3.2</a></b>	<b>CHECHER DE L'AIDE SUR LES CLASSES AVEC VS.NET.....</b>	<b>77</b>
<a href="#">3.2.1</a>	OPTION AIDE.....	77
<a href="#">3.2.2</a>	AIDE/INDEX.....	79
<b><a href="#">3.3</a></b>	<b>LA CLASSE STRING.....</b>	<b>80</b>
<b><a href="#">3.4</a></b>	<b>LA CLASSE ARRAY.....</b>	<b>82</b>
<b><a href="#">3.5</a></b>	<b>LA CLASSE ARRAYLIST.....</b>	<b>84</b>
<b><a href="#">3.6</a></b>	<b>LA CLASSE HASHTABLE.....</b>	<b>86</b>
<b><a href="#">3.7</a></b>	<b>LA CLASSE STREAMREADER.....</b>	<b>89</b>
<b><a href="#">3.8</a></b>	<b>LA CLASSE STREAMWRITER.....</b>	<b>90</b>
<b><a href="#">3.9</a></b>	<b>LA CLASSE REGEX.....</b>	<b>91</b>
<a href="#">3.9.1</a>	VÉRIFIER QU'UNE CHAÎNE CORRESPOND À UN MODÈLE DONNÉ.....	93
<a href="#">3.9.2</a>	TROUVER TOUS LES ÉLÉMENTS D'UNE CHAÎNE CORRESPONDANT À UN MODÈLE.....	94
<a href="#">3.9.3</a>	RÉCUPÉRER DES PARTIES D'UN MODÈLE.....	95
<a href="#">3.9.4</a>	UN PROGRAMME D'APPRENTISSAGE.....	96
<a href="#">3.9.5</a>	LA MÉTHODE SPLIT.....	97
<b><a href="#">3.10</a></b>	<b>LES CLASSES BINARYREADER ET BINARYWRITER.....</b>	<b>98</b>

### **[4. INTERFACES GRAPHIQUES AVEC VB.NET ET VS.NET.....](#)** **102**

<b><a href="#">4.1</a></b>	<b>LES BASES DES INTERFACES GRAPHIQUES.....</b>	<b>102</b>
<a href="#">4.1.1</a>	UNE FENÊTRE SIMPLE.....	102
<a href="#">4.1.2</a>	UN FORMULAIRE AVEC BOUTON.....	103
<b><a href="#">4.2</a></b>	<b>CONSTRUIRE UNE INTERFACE GRAPHIQUE AVEC VISUAL STUDIO.NET.....</b>	<b>106</b>
<a href="#">4.2.1</a>	CRÉATION INITIALE DU PROJET.....	106
<a href="#">4.2.2</a>	LES FENÊTRE DE L'INTERFACE DE VS.NET.....	107
<a href="#">4.2.3</a>	EXÉCUTION D'UN PROJET.....	109
<a href="#">4.2.4</a>	LE CODE GÉNÉRÉ PAR VS.NET.....	109
<a href="#">4.2.5</a>	COMPILATION DANS UNE FENÊTRE DOS.....	111
<a href="#">4.2.6</a>	GESTION DES ÉVÉNEMENTS.....	112
<a href="#">4.2.7</a>	CONCLUSION.....	112
<b><a href="#">4.3</a></b>	<b>FENÊTRE AVEC CHAMP DE SAISIE, BOUTON ET LIBELLÉ.....</b>	<b>112</b>
<a href="#">4.3.1</a>	CONCEPTION GRAPHIQUE.....	112
<a href="#">4.3.2</a>	GESTION DES ÉVÉNEMENTS D'UN FORMULAIRE.....	115
<a href="#">4.3.3</a>	UNE AUTRE MÉTHODE POUR GÉRER LES ÉVÉNEMENTS.....	117
<a href="#">4.3.4</a>	CONCLUSION.....	119
<b><a href="#">4.4</a></b>	<b>QUELQUES COMPOSANTS UTILES.....</b>	<b>119</b>
<a href="#">4.4.1</a>	FORMULAIRE FORM.....	119
<a href="#">4.4.2</a>	ÉTIQUETTES LABEL ET BOÎTES DE SAISIE TEXTBOX.....	120
<a href="#">4.4.3</a>	LISTES DÉROULANTES COMBOBOX.....	121
<a href="#">4.4.4</a>	COMPOSANT LISTBOX.....	123
<a href="#">4.4.5</a>	CASES À COCHER CHECKBOX, BOUTONS RADIO BUTTONRADIO.....	125

<a href="#">4.4.6</a>	VARIATEURS SCROLLBAR.....	126
<a href="#">4.5</a>	ÉVÉNEMENTS SOURIS.....	128
<a href="#">4.6</a>	CRÉER UNE FENÊTRE AVEC MENU.....	130
<a href="#">4.7</a>	COMPOSANTS NON VISUELS.....	134
<a href="#">4.7.1</a>	BOÎTES DE DIALOGUE OPENFILEDIALOG ET SAVEFILEDIALOG.....	134
<a href="#">4.7.2</a>	BOÎTES DE DIALOGUE FONTCOLOR ET COLORDIALOG.....	138
<a href="#">4.7.3</a>	TIMER.....	140
<a href="#">4.8</a>	L'EXEMPLE IMPOTS.....	142
<b><a href="#">5.</a></b>	<b>GESTION D'ÉVÉNEMENTS.....</b>	<b>147</b>
<a href="#">5.1</a>	OBJETS DELEGATE.....	147
<a href="#">5.2</a>	GESTION D'ÉVÉNEMENTS.....	148
<a href="#">5.2.1</a>	DÉCLARATION D'UN ÉVÉNEMENT.....	148
<a href="#">5.2.2</a>	DÉFINIR LES GESTIONNAIRES D'UN ÉVÉNEMENT.....	148
<a href="#">5.2.3</a>	DÉCLENCHER UN ÉVÉNEMENT.....	148
<a href="#">5.2.4</a>	UN EXEMPLE.....	149
<b><a href="#">6.</a></b>	<b>ACCÈS AUX BASES DE DONNÉES.....</b>	<b>153</b>
<a href="#">6.1</a>	GÉNÉRALITÉS.....	153
<a href="#">6.2</a>	LES DEUX MODES D'EXPLOITATION D'UNE SOURCE DE DONNÉES.....	154
<a href="#">6.3</a>	ACCÈS AUX DONNÉES EN MODE CONNECTÉ.....	155
<a href="#">6.3.1</a>	LES BASES DE DONNÉES DE L'EXEMPLE.....	155
<a href="#">6.3.2</a>	UTILISATION D'UN PILOTE ODBC.....	159
<a href="#">6.3.2.1</a>	La phase de connexion.....	160
<a href="#">6.3.2.2</a>	Émettre des requêtes SQL.....	161
<a href="#">6.3.2.3</a>	Exploitation du résultat d'une requête SELECT.....	162
<a href="#">6.3.2.4</a>	Libération des ressources.....	163
<a href="#">6.3.3</a>	UTILISATION D'UN PILOTE OLE DB.....	163
<a href="#">6.3.4</a>	MISE À JOUR D'UNE TABLE.....	164
<a href="#">6.3.5</a>	IMPOTS.....	168
<a href="#">6.4</a>	ACCÈS AUX DONNÉES EN MODE DÉCONNECTÉ.....	172
<b><a href="#">7.</a></b>	<b>LES THREADS D'EXÉCUTION.....</b>	<b>173</b>
<a href="#">7.1</a>	INTRODUCTION.....	173
<a href="#">7.2</a>	CRÉATION DE THREADS D'EXÉCUTION.....	174
<a href="#">7.3</a>	INTÉRÊT DES THREADS.....	176
<a href="#">7.4</a>	ACCÈS À DES RESSOURCES PARTAGÉES.....	177
<a href="#">7.5</a>	ACCÈS EXCLUSIF À UNE RESSOURCE PARTAGÉE.....	178
<a href="#">7.6</a>	SYNCHRONISATION PAR ÉVÉNEMENTS.....	181
<b><a href="#">8.</a></b>	<b>PROGRAMMATION TCP-IP.....</b>	<b>184</b>
<a href="#">8.1</a>	GÉNÉRALITÉS.....	184
<a href="#">8.1.1</a>	LES PROTOCOLES DE L'INTERNET.....	184
<a href="#">8.1.2</a>	LE MODÈLE OSI.....	184
<a href="#">8.1.3</a>	LE MODÈLE TCP/IP.....	185
<a href="#">8.1.4</a>	FONCTIONNEMENT DES PROTOCOLES DE L'INTERNET.....	187
<a href="#">8.1.5</a>	L'ADRESSAGE DANS L'INTERNET.....	188
<a href="#">8.1.5.1</a>	Les classes d'adresses IP.....	189
<a href="#">8.1.5.2</a>	Les protocoles de conversion Adresse Internet <--> Adresse physique.....	190
<a href="#">8.1.6</a>	LA COUCHE RÉSEAU DITE COUCHE IP DE L'INTERNET.....	190
<a href="#">8.1.6.1</a>	Le routage.....	191
<a href="#">8.1.6.2</a>	Messages d'erreur et de contrôle.....	191
<a href="#">8.1.7</a>	LA COUCHE TRANSPORT : LES PROTOCOLES UDP ET TCP.....	192
<a href="#">8.1.7.1</a>	Le protocole UDP : User Datagram Protocol.....	192
<a href="#">8.1.7.2</a>	Le protocole TCP : Transfer Control Protocol.....	192

<a href="#">8.1.8</a>	LA COUCHE APPLICATIONS.....	192
<a href="#">8.1.9</a>	CONCLUSION.....	193
<b><a href="#">8.2</a></b>	<b>GESTION DES ADRESSES RÉSEAU.....</b>	<b>194</b>
<b><a href="#">8.3</a></b>	<b>PROGRAMMATION TCP-IP.....</b>	<b>196</b>
<a href="#">8.3.1</a>	GÉNÉRALITÉS.....	196
<a href="#">8.3.2</a>	LES CARACTÉRISTIQUES DU PROTOCOLE TCP.....	197
<a href="#">8.3.3</a>	LA RELATION CLIENT-SERVEUR.....	197
<a href="#">8.3.4</a>	ARCHITECTURE D'UN CLIENT.....	197
<a href="#">8.3.5</a>	ARCHITECTURE D'UN SERVEUR.....	198
<a href="#">8.3.6</a>	LA CLASSE TcpCLIENT.....	198
<a href="#">8.3.7</a>	LA CLASSE NETWORKSTREAM.....	198
<a href="#">8.3.8</a>	ARCHITECTURE DE BASE D'UN CLIENT INTERNET.....	199
<a href="#">8.3.9</a>	LA CLASSE TcpLISTENER.....	199
<a href="#">8.3.10</a>	ARCHITECTURE DE BASE D'UN SERVEUR INTERNET.....	200
<b><a href="#">8.4</a></b>	<b>EXEMPLES.....</b>	<b>200</b>
<a href="#">8.4.1</a>	SERVEUR D'ÉCHO.....	200
<a href="#">8.4.2</a>	UN CLIENT POUR LE SERVEUR D'ÉCHO.....	202
<a href="#">8.4.3</a>	UN CLIENT TCP GÉNÉRIQUE.....	204
<a href="#">8.4.4</a>	UN SERVEUR TCP GÉNÉRIQUE.....	209
<a href="#">8.4.5</a>	UN CLIENT WEB.....	215
<a href="#">8.4.6</a>	CLIENT WEB GÉRANT LES REDIRECTIONS.....	217
<a href="#">8.4.7</a>	SERVEUR DE CALCUL D'IMPÔTS.....	219
<b><a href="#">9.</a></b>	<b>SERVICES WEB.....</b>	<b>225</b>
<b><a href="#">9.1</a></b>	<b>INTRODUCTION.....</b>	<b>225</b>
<b><a href="#">9.2</a></b>	<b>LES NAVIGATEURS ET XML.....</b>	<b>225</b>
<b><a href="#">9.3</a></b>	<b>UN PREMIER SERVICE WEB.....</b>	<b>226</b>
<a href="#">9.3.1</a>	VERSION 1.....	226
<a href="#">9.3.2</a>	VERSION 2.....	232
<a href="#">9.3.3</a>	VERSION 3.....	234
<a href="#">9.3.4</a>	VERSION 4.....	235
<a href="#">9.3.5</a>	CONCLUSION.....	236
<b><a href="#">9.4</a></b>	<b>UN SERVICE WEB D'OPÉRATIONS.....</b>	<b>236</b>
<b><a href="#">9.5</a></b>	<b>UN CLIENT HTTP-POST.....</b>	<b>241</b>
<b><a href="#">9.6</a></b>	<b>UN CLIENT SOAP.....</b>	<b>250</b>
<b><a href="#">9.7</a></b>	<b>ENCAPSULATION DES ÉCHANGES CLIENT-SERVEUR.....</b>	<b>254</b>
<a href="#">9.7.1</a>	LA CLASSE D'ENCAPSULATION.....	254
<a href="#">9.7.2</a>	UN CLIENT CONSOLE.....	257
<a href="#">9.7.3</a>	UN CLIENT GRAPHIQUE WINDOWS.....	260
<b><a href="#">9.8</a></b>	<b>UN CLIENT PROXY.....</b>	<b>264</b>
<b><a href="#">9.9</a></b>	<b>CONFIGURER UN SERVICE WEB.....</b>	<b>268</b>
<b><a href="#">9.10</a></b>	<b>LE SERVICE WEB IMPOTS.....</b>	<b>270</b>
<a href="#">9.10.1</a>	LE SERVICE WEB.....	270
<a href="#">9.10.2</a>	GÉNÉRER LE PROXY DU SERVICE IMPOTS.....	276
<a href="#">9.10.3</a>	UTILISER LE PROXY AVEC UN CLIENT.....	277
<b><a href="#">10.</a></b>	<b>A SUIVRE.....</b>	<b>281</b>