Programmation événementielle avec VB.NET

Manel ZOGHLAMI

ISETJ – Tunisie

2012

Chapitre 1. La syntaxe et les éléments de bases de VB.NET

VB.net est un langage de programmation orienté objet, très utilisé dans la programmation événementielle. VB est l'abréviation de VISUAL BASIC:

- **BASIC** → Dérivé du langage **basic** (**B**eginner's **A**ll purpose **S**ymbolic **I**nstruction **C**ode) des années 60
- **VISUAL** → Dessin et aperçu de l'interface avant l'exécution

Dans ce chapitre on ne va pas s'intéresser à l'aspect événementiel, on va plutôt se contenter sur la syntaxe et les structures de base du langage VB.net.

1. Les règles de base

- VB n'est pas sensible à la casse (pas comme d'autres langages comme Java et C) : il ne fait pas la différence entre majuscule et minuscule.

Exemple: La variable nommée id, celle nommée ID ou iD représentent la même variable en VB.

- Le commentaire commence par le caractère suivant '
- Chaque instruction VB est écrite généralement dans une ligne à part : la fin de la ligne représente la fin de l'instruction. Elle ne finit pas par un ; comme en C ou Java.

Exemple: `Dim age As Integer` est une instruction qui déclare un entier nommé age.

Cependant, Plusieurs instructions peuvent se suivre sur une même ligne, séparées par ':'

Exemple:

`Dim age As Integer : Dim nom As String` est equivalent à

```
Dim age As Integer
Dim nom As String
```

- L'indentation est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

2. Les variables

2.1. Les types de données

Le langage VB.NET utilise **plusieurs types** de données. Ceux qui sont couramment utilisés sont résumés dans le tableau ci-dessous.

Type Visual Basic	Allocation de stockage nominal	Plage de valeurs
Byte	1 octet	0 à 255 (non signé)
Short (entier court)	2 octets	(De -32 768 à 32 767 (signé))
Integer (entier)	4 octets	-2 147 483 648 à 2 147 483 647 (signé)
Long (entier long)	8 octets	
Single (nombre à virgule flottante simple précision)	4 octets	
Double (nombre à virgule flottante double précision)	8 octets	
Boolean	Dépend de la plateforme d'implémentation	True ou False
Char (caractère unique)	2 octets	0 à 65 535 (non signé)
Date	8 octets	Du 1er janvier 0001 0:00:00 (minuit) au 31 décembre 9999 23:59:59
Object	4 octets sur une plateforme 32 bits 8 octets sur une plateforme 64 bits	N'importe quel type peut être stocké dans une variable de type Object
String (longueur variable)	Dépend de la plateforme d'implémentation	

La déclaration se fait avec le mot réservé ***Dim:*** `Dim NomVariable As Type`

Exemples:

```
Dim Taux As Single ‘ Taux est décimal
```

Dim Reponse As String ‘ Mot proposé par l'utilisateur

Par défaut, le compilateur VB considère que toute variable qui apparaît doit avoir été déclarée. Toutefois, si on ajoute la ligne **Option Explicit Off ,** VB sera **permissif** : il nous permet d'utiliser des variables sans les déclarer. Ceci n'est pas recommandé, il est préférable de laisser la valeur par défaut et toujours déclarer les variables.

Par défaut : Option Explicit on

Exemple:

```
Option Explicit On
'valeur par défaut, on peut donc ne
'pas écrire cette ligne

Module Module1

    Sub Main()
        Dim i As Integer = 3
        s = i
        Console.Out.WriteLine(s)

    End Sub

End Module
```

➔ Erreur: s n'est pas déclaré

```
Option Explicit off
' ceci n est pas recommandé

Module Module1

    Sub Main()
        Dim i As Integer = 3
        s = i
        Console.Out.WriteLine(s)

    End Sub

End Module
```

➔ Pas d'erreur, on affiche 3

2.2. Les tableaux

2.2.1. Tableau à une dimension

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

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

n est l'indice du dernier élément de tableau. La syntaxe nomTableau (i) désigne la donnée de la case numéro i avec $0 \leq i \leq n$. Si i n'appartient pas à l'intervalle [0,n], on va avoir une erreur (appelée exception en VB).

On peut initialiser un tableau lors de sa déclaration. Dans ce cas, on n'a pas besoin d'indiquer sa taille :

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

Les tableaux ont une propriété Length qui est le nombre d'éléments du tableau. Cette propriété est surtout utilisée pour parcourir les éléments d'un tableau.

Exemple:

```
Module TesterTableau
    Sub Main()
        Dim Tab(5) As Integer

        'remplir le tableau
        For i = 0 To Tab.Length - 1
            Tab(i) = i * 10
        Next

        'affichage
        Console.Out.WriteLine("Taille du tableau: " & Tab.Length)
        For i As Integer = 0 To Tab.Length - 1

            Console.Out.WriteLine("Tab(" & i & ")=" & Tab(i))
        Next

    End Sub

End Module
```

Exécution

Taille du tableau: 6
Tab(0)= 0
Tab(1)= 10
Tab(2)= 20
Tab(3)= 30
Tab(4)= 40
Tab(5)= 50

Tab	0	10	20	30	40	50
	0	1	2	3	4	5

Exercice :

Ecrire le code VB.NET permettant de demander les 3 matières le plus préférées pour un étudiant, les stocker dans un tableau puis les afficher.

Solution:

```
        Dim TabMatiere(2) As String

         For i = 0 To TabMatiere.Length - 1
             Console.Out.WriteLine("matière préférée n°" & i + 1)
             TabMatiere(i) = Console.In.ReadLine
         Next

         Console.Out.WriteLine("Vos matières préférées sont: ")
         For i As Integer = 0 To TabMatiere.Length - 1
             Console.Out.WriteLine(i + 1 & ": " & TabMatiere(i))
         Next
```

2.2.2. Tableau à plusieurs dimensions:

On peut déclarer des tableaux de deux, trois, quatre, dimensions ou plus. Un tableau à deux dimensions est déclaré comme suit :

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

ou 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.

On peut initialiser le tableau à deux dimensions comme suit :

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

Le nombre de éléments dans chaque dimension peut être obtenue en utilisant la méthode **GetLenth(i)** ou i=0 représente la dimension correspondant au 1er indice, i=1 la dimension correspondant au 2ième indice, …Voici un programme d'exemple :

```
Dim tab( 2, 1) As Integer       '3 lignes et 2 colonnes
For i As Integer = 0 To tab.GetLength(0) - 1      'i varit de 0à2
   For j As Integer = 0 To tab.GetLength(1) - 1   'j varit de 0à1
            tab(i, j) = i * 10 + j
   Next
Next

For i As Integer = 0 To tab.GetLength(0) - 1
    For j As Integer = 0 To tab.GetLength(1) - 1

      Console.Out.WriteLine("tab("& i&","& j&")=" & tab(i, j))
    Next
Next
```

Execution:
tab(0,0)= 0
tab (0,1)= 1

```
tab (1,0)= 10
tab (1,1)= 11
tab (2,0)= 20
tab (2,1)= 21
```

Ligne 0	0	1
Ligne 1	10	11
Ligne 2	20	21

2.3. Les conversions entre types:

2.3.1. Conversion entre nombres et chaines de caractères

Par défaut, la conversion implicite est permise. Cependant, la directive [**Option Strict on**] interdit toute conversion de types de données pouvant entrainer une perte de données et toute conversion entre les types numériques et les chaines. Il faut alors explicitement utiliser des fonctions de conversion.

Remarque: Par défaut, on a [**Option Strict off**] 'conversion implicite permise

Les principales fonctions de conversion sont:

a) Nombre vers chaines :
nombre.ToString ou "" & nombre ou CType(nombre,String) ou CStr(nombre)
b) Parse (chaine)

chaine → Integer	Integer.Parse(chaine) ou Int32.Parse
chaîne → Long	Long.Parse(chaine) ou Int64.Parse
chaîne → Double	Double.Parse(chaîne)
chaîne → Single	Single.Parse(chaîne)

L'exemple ci-dessous porte sur la conversion entre chaines et nombres et l'impact de la directive Option Strict

```
Option Strict off
' par défaut donc on peut
' supprimer cette ligne
Module Module1

    Sub Main()
        ' nombre --> chaîne
        Dim i As Integer = 3
        Dim s As String
        s = i
        Console.Out.WriteLine(s)

        ' chaîne --> nombre
        Dim i2 As Integer
        Dim s2 As String = "4"
        i2 = s2
        Console.Out.WriteLine(i2)

        Console.In.ReadLine()

    End Sub
End Module
```

Résultat:

3
4

```
Option Strict On
Module Module1

    Sub Main()
        ' nombre --> chaîne
        Dim i As Integer = 3
        Dim s As String
        s = i
        Console.Out.WriteLine(s)


        ' chaîne --> nombre
        Dim i2 As Integer
        Dim s2 As String = "4"
        i2 = s2
        Console.Out.WriteLine(i2)

        Console.In.ReadLine()

    End Sub

End Module
```

- ➔ Erreur 1: Option Strict On interdit les conversions implicites de 'Integer' en 'String'.
- ➔ Erreur 2 Option Strict On interdit les conversions implicites de 'String' en 'Integer'.

Solution:

```
S= CType(i, String) ou bien
s = i.ToString ou bien s= ""& i
ou bien s = CStr(i)
i2=Integer.Parse(s2)
```

Résultat:

3
4

Il est à noter que les nombres réels sous forme de chaîne de caractères doivent utiliser la virgule et non le point décimal. Ainsi on écrit

Dim d As Double = 3.1 et Dim d2 As Double = Double.Parse("4,2")

c) **Ctype:** On peut utiliser la fonction *CType(expression, type)* qui permet de convertir l'expression vers le type précisé. Le programme ci-dessous illustre l'utilisation de cette fonction.

```
Module CtypeModule
    Sub main()
        Dim var1 As Date = CType("20 janvier 2011", Date)
        Dim var2 As Long = CType("333", Long)
        Dim var3 As Boolean = CType("true", Boolean)
        Dim var4 As Byte = CType("222", Byte)
        Dim var5 As Char = CType("A", Char)
        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)

    End Sub
End Module
```

Execution:

20/01/2011
333
True
222
A

d) Objet vers Chaine : objet.ToString : une représentation textuelle d'un objet

e) CBool,CByte,CChar,CDate,CDbl,CDec,CInt,CLng,CObj,CShort,CSng,CStr
Le résultat de ces fonctions est :

- CBool : boolean
- Cbyte : byte
- CChar : Char
- CDate: date
- CDbl :double
- CDec : decimal
- CInt : Integer
- CLng: long
- CObj: object
- CShort : short
- CSng: single

- CStr: String

```
Module CXModule
    Sub main()
        Dim var1 As Date = CDate("20 janvier 2011")
        Dim var2 As Long = CLng("333")
        Dim var3 As Boolean = CBool("true")
        Dim var4 As Byte = CByte("222")
        Dim var5 As Char = CChar("A")
        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)

    End Sub
End Module
```

Execution:

20/01/2011
333
True
222
A

Il est à noter que 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 qu'on appelle *exception* en VB.NET. La gestion de ce genre d'erreurs va être étudiée plus.

3. Les opérateurs

3.1. Opérateurs arithmétiques

VB reconnaît les opérateurs arithmétiques usuels qui sont résumés dans le tableau suivant :

Opérateur	*Description*	Exemples
+, -	Addition et soustraction	
*	Multiplication	
/	Division décimale	5 / 2 = 2.5
^	Puissance	5^ 2 =25
\	Division entière	5 \ 2 = 2

MOD	Modulo (reste de la division entière)	5 MOD 2 = 1

Il existe diverses fonctions mathématiques qui sont définies dans une classe .NET appelée Math. Voici quelques-unes:

- Sqrt (x) racine carree
- Pow(*x, y*) x à la puissance y (x>0)
- Abs(x) valeur absolue
- Cos(x) Cosinus
- Sin(x) Sinus

Exemple:
Dim a, b As Double
a = Math.Sqrt(4)

b = Math.Cos(2)

3.2. Opérateurs de comparaison

Le résultat d'une expression de comparaison est un booléen. Les opérateurs sont les suivants:

- = : 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.

● Comparaison des caractères

Soient deux caracteres C1 et C2. Il est possible de les comparer avec les operateurs : <, <=, =, <>, >, >=. 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'

```
Module ComparerStrings
    Sub main()
```

```
        Dim aMaj As Char = "A"
        Dim bMin As Char = "b"
        Dim aMin As Char = "a"
        Console.Out.WriteLine("a<b=" & (aMin < bMin))
        Console.Out.WriteLine("A<a=" & (aMaj < aMin))

    End Sub

End Module
```

Exécution:

a<b= true

A<a = true

- **Comparaison des chaines de caractères**

Les chaines de caractères sont comparées caractère par caractère. La première inégalité rencontrée entre deux caractères induit une inégalité de même sens sur les chaines.

Exemple:

```
Module ComparerStrings
    Sub main()
        Dim exemple As String = "exemple"
        Dim example As String = "example"
        Dim MajExemple As String = "EXEMPLE"
        Dim exe As String = "exe"
        Console.Out.WriteLine("exemple<example=" & (exemple < example))
        Console.Out.WriteLine("exemple < EXEMPLE =" & (exemple <
MajExemple))
        Console.Out.WriteLine("exe< exemple =" & (exe < exemple))
        Console.Out.WriteLine("exemple like exe*=" & ("exemple" Like
"exe*"))

    End Sub
End Module
```

Exécution:

exemple<example= False
exemple < EXEMPLE = False
exe< exemple = True
exemple like exe*= True

3.3. Opérateurs logiques

Les Opérations logiques ou opérateurs de bits sont les suivants:

- And : 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 operateurs entre-eux est la suivante:

1. Not
2. And, AndAlso
3. Or, OrElse
4. Xor

3.4. Opérateurs associé à une affectation

Il est possible d'écrire a+=b qui signifie a=a+b.

Exemple:

```
Dim a As Integer = 3
Dim b As Integer = 2
a += b
Console.Out.WriteLine(a)
```

➔ On affiche 5

De même on a:

- a- =b → a=a-b
- a*=b → a=a*b
- a/=b → a=a/b
- a\=b → a=a\b
- a^=b →a=a^b
- a&=b → a=a&b (ici a et b sont des chaines de caractères)

4. Les sous-programmes

VB .NET permet l'utilisation des procédures et des fonctions avec ou sans paramètres. On rappelle que la différence principale entre la procédure et la fonction est que cette dernière retourne une valeur lorsqu'elle est appelée.

4.1. Les procédures

La syntaxe est la suivante:

```
Sub NomProcédure (paramètres)
  ' Instructions
```

```
End Sub
```

Pour appeler une procédure, on écrit:

Call NomProcédure (paramètre1, …, paramètreN)

NomProcédure paramètre1, …, paramètreN

Exemple

Call Remplissage(n, T)

Remplissage n, T

4.2. Les fonctions

La syntaxe est la suivante:

```
function NomFonction (paramètres) As type
        ' Instructions
        NomFonction = Valeur_retour
End Function
```

Si aucun type n'est spécifié, la fonction est de type *Variant*.

L'appel de la fonction est de la forme :

variable = ***NomFonction*** (paramètre1, …, paramètreN)

Exemple

max = ValeurMaximale (n, T)

4.3. Passage des paramètres

Il existe deux modes de Passage des paramètres :

Par valeur (ByVal) : le contenu de la variable ne sera pas modifié en sortie du sous-programme. On travaille sur une copie de la variable passée en paramètre.

Par référence (ByRef) : le contenu de la variable pourra être modifié en sortie de sous-programme. On travaille sur l'adresse de la variable passée en paramètre.

Les paramètres d'une fonction/procédure sont par défaut passés par valeur : c'est à dire que la valeur du paramètre effectif est recopiée dans le paramètre formel correspondant.

4.4. Structure générale d'un programme

Une application VB peut être composée d'un ou de plusieurs formulaires et d'un ou de plusieurs modules. Dans chaque module ou formulaire, des variables, des procédures et/ou des fonctions peuvent être déclarées.

On peut ne pas démarrer le projet par un formulaire mais par une procédure, particulièrement dans le cas des programmes sans interface graphique. La procédure de démarrage doit donc obligatoirement se trouver dans un **module** et doit porter le nom **Main**().

Remarque: Dans le menu *Propriétés* de *projet*, on sélectionne l'objet de démarrage de notre application.

La structure générale d'un module contenant la procédure main est la suivante:

```
' options
Option Explicit On
Option Strict On

' espaces de noms
Imports espace1
Imports ....

Module nomDuModule

 Sub main()
   ....
 End Sub

        ' Fonctions et procédures
Sub nomProcdure(… )
          ……..
         End Sub

Function nomFonction(… )
          ……..
         End Function

        End Module
```

Exemple: Calcul du factoriel d'un entier n

Donner un programme qui utilise une procédure Factoriel qui prend en paramètre la valeur de n et la variable résultat fact pour calculer le factoriel d'un entier.

```
Module calculFact

 Sub Factoriel(ByVal n As Integer, ByRef Fact As Integer)
        Dim i As Integer
        For i = 1 To n
            Fact = Fact * i
        Next
    End Sub

    Sub main()
        Dim fact As Integer = 1
        Factoriel(3, fact)
        Console.WriteLine(fact)
        Console.ReadLine()
    End Sub
End Module
```

Execution:

6

n est passée par valeur (utilisée seulement pour calculer Fact) alors que Fact doit être passée par référence car son contenu change.

Si on passe fact par valeur, on aura 1 comme résultat: Fact n'est pas influencée par la procédure et a conservé sa valeur d'appel.

Exercice:

Donner une autre solution en utilisant une fonction.

Solution:

```
Module calculFact
    Function Factoriel(ByVal n As Integer) As Integer
        Dim Fact As Integer = 1
        Dim i As Integer
        For i = 1 To n
            Fact = Fact * i
        Next
        Factoriel = Fact
    End Function

    Sub main()
        Dim res As Integer = Factoriel(3)
        Console.WriteLine(res)
        Console.ReadLine()
    End Sub
End Module
```

Les deux instructions ***Exit function*** et ***Exit Sub*** permettent respectivement l'interruption d'une fonction/procédure, sans exécution des instructions restante.

Remarque: Dans chaque procédure et fonction, on peut déclarer des variables locales. Une fonction ou une procédure peut être déclarée soit *Privée (Private),* soit *Publique (Public).* Le sens de *Privé ou Public* est relatif au formulaire ou au module dans lesquelles elles sont déclarées.

5. Les structures de contrôle

5.1. Les structures conditionnelles

Le programmeur est très souvent amené à **tester** des valeurs et à orienter le programme selon ces valeurs.

5.1.1. Structure conditionnelle si .. alors

La syntaxe en VB est :

```
if condition then
        instructions_alors
else
```

```
        instructions_sinon
end if
```

La clause else est optionnelle.

L'instruction end if est nécessaire meme s'il s'agit d'une seule instruction dans le bloc if.

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

```
Module Condition_Module
    Sub main()
        Dim a As Integer = 3
        If a > 5 Then
            Console.Out.WriteLine(a & " est > " & 5)
        Else
            If a = 5 Then
                Console.Out.WriteLine(a & " est = " & 5)
            Else
                Console.Out.WriteLine(a & " est < " & 5)
            End If
        End If


    End Sub
End Module
```

5.1.2. Structure conditionnelle à alternatives: selon … faire

L'instruction ***Select Case*** est une instruction conditionnelle à *alternatives,* c'est-à-dire qu'une *expression* peut être testée par rapport à plusieurs valeurs possibles. La syntaxe est la suivante:

```
select case expression
case liste_valeurs1
instructions1
case liste_valeurs2
instructions2
...
case else
instructions_sinon
end select
```

La clause [case else] est optionnelle.

Les instructions se trouvant après *'Case Liste_Valeurs_i'* seront exécutées si *'expression = à l'un des éléments de Liste_Valeurs_i', i = 1, 2, 3, ….* Sinon, les

instructions se trouvant après *'Else Case'* seront exécutées. seules les actions liées à la première condition vérifiée sont exécutées.

Le type de [expression] doit etre l'un des types suivants :
Integer, Long, Short, Single, String , Boolean ,Object, Byte, Char, Date, Decimal, Double.

*Liste_Valeurs*_i sont des valeurs possibles de l'expression. Elle peut être :

- Une valeur : 2
- une suite de valeurs : 1, 3, 5, 7, 9
- une fourchette de valeur : 0 To 9
- une plage de valeurs : **Is** >= 10 (Is est un mot réservé optionnel)
- idem avec les opérateurs =, <, <=, >, >=, < >

Notez que *'Liste_Valeurs_i'* peut être une combinaison de listes de valeurs comme dans le cas des exemples suivants :

- Case 1 To 4, 7 To 9, 11, 13, Is > NombreMAx
- Case "Lundi", "Mercredi", "Dimanche", VariableJour

Exemple:

```
Dim a As Integer = 22
        Select Case a
            Case 1 To 6, 8 To 10
                Console.Out.WriteLine("a est dans l'intervalle [1,6] ou
[8,10]")
            Case Is > 12
                Console.Out.WriteLine("a est > 12 ")
            Case Else
                Console.Out.WriteLine("a est egal à  7 ,11 ou 12")

        End Select
```

5.1.3. Autres structures conditionnelles

Il existe d'autres structures conditionnelles qui remplacent les structures conditionnelles connues déjà citées.

- **IIf :** On peut utiliser le mot clé IIF avec la syntaxe suivante:

IIf (Condition, ValeurSiVrai, ValeurSiFaux).

Ceci permet d'évaluer une expression et, selon la valeur logique prise par cette expression, on affecte une valeur ou une autre à une variable. Cette structure peut remplacer un bloc If … Then … Else … End If . L'exemple suivant illustre une utilisation possible de IIF:

```
Dim Note As Single
Dim Réponse As String
Note = 15
Réponse = IIf(Note >= 10, " Admis ", " Ajourné ")

Console.Out.WriteLine(Réponse)
```

Exécution: Admis

Exercice

Donner la version équivalente à l'exemple précédent en utilisant un bloc IF .

Réponse

```
Dim Note As Single
Dim Reponse As String
Note = 15
If Note >= 10 Then
    Reponse = "Admis"
Else
    Reponse = "Ajourné"
End If

Console.Out.WriteLine(Reponse)
```

Exécution: Admis

Choose: Elle Renvoie une valeur parmi une liste de valeurs en fonction d'un indice. Cette structure peut remplacer le bloc If ainsi que la structure SelectCase. La syntaxeest la suivante:

Variable = Choose (indexe , valeur1, valeur2, … , valeurN)

L'indexe est de type double. Il doit varier entre 1 et le nombre de valeurs (N)

```
Dim couleur As String
Dim numCouleur As Integer
numCouleur=3
Couleur = Choose(numCouleur, "Rouge", "Vert", "Bleu")
Console.Out.WriteLine(couleur)
```

Exécution:

Bleu

Exercice

Donner la version équivalente à l'exemple précédent en utilisant un bloc Select Case.

Réponse

```
Dim couleur As String
Dim numCouleur As Integer
numCouleur = 3
Select Case numCouleur
    Case 1
        couleur = "Rouge"
    Case 2
        couleur = "Vert"
    Case 3
        couleur = "Bleu"

End Select
Console.Out.WriteLine(couleur)
```

5.2. Structures de contrôle itératives

Une boucle sert à exécuter un ensemble d'instructions répétées plusieurs fois. On peut sortir d'une boucle suite à une condition ou suite au mot clé *Exit*.

5.2.1. La boucle pour

Elle est utilisée si le Nombre de répétitions est connu à l'avance. La syntaxe est la suivante:

For compteur [**as** type] = val_début ***To*** val_fin ***[Step*** pas ***]***
    instructions
Next [compteur]

Par défaut, le type du compteur est entier, le pas est égal à 1.

Exemple:

```
Console.Out.WriteLine("Factoriel 3")
Dim n As Integer = 3
Dim fact As Integer = 1
For i = 1 To n
    fact = fact * i
Next i
Console.Out.WriteLine(fact)

L'exécution donne:
6
```

```
Console.Out.WriteLine("Factoriel
(3)")
Dim n As Integer = 3
Dim fact As Integer = 1
For i = n To 1 Step -1
     fact = fact * i
Next i
Console.Out.WriteLine(fact)
```

On peut interrompre la boucle (sans exécution des instructions restantes) en utilisant le mot clé: *Exit For*

Exemple:

```
Dim n As Integer = 8
For i = 0 To n
    If i = 3 Then
        Exit For
    End If
    Console.WriteLine(i)
Next i
```

Execution:

0
1
2

Une autre forme de boucle peut être utilisée quand on connait le nombre d'itérations à l'avance, elle a la syntaxe suivante:

For each element [***as*** type] ***in*** groupe
instructions
Next [element]

groupe est une collection d'objets. La collection d'objets déjà vue est le tableau

type est le type des objets de la collection. Pour un tableau, il s'agit du type des éléments du tableau

element est une variable locale à la boucle qui va prendre pour valeur successivement toutes les valeurs de la collection.

Exemple:

```
Dim langages() As String = {"JAVA", "VB", "C", "Pascal"}
For Each lgg As String In langages
    Console.Out.WriteLine(lgg)
Next
```

Exercice:

Afficher la somme des éléments d'un tableau d'entier par deux méthodes: boucle For et boucle ForEach.

Solution:

```
Dim tab() As Integer = {3, 4, 7}
Dim somme As Integer = 0
For i = 0 To tab.Length - 1
     somme = somme + tab(i)
Next

Console.WriteLine(somme)
```

```
Dim tab() As Integer = {3, 4, 7}
Dim somme As Integer = 0
For Each ele As Integer In tab
      somme = somme + ele

Next
Console.WriteLine(somme)
```

5.2.2. La boucle tant que

On boucle tant que la condition est vérifiée.

Do While (condition) instructions ***Loop***	***While*** (condition) instructions ***End While***	***Do*** instructions ***Loop While (***condition***)***
➔ On teste la condition au début de la boucle, donc il y a une possibilité de ne pas entrer dans la boucle : instructions exécutées 0 ou plusieurs fois.		➔ On teste la condition à la fin: instructions exécutées au moins une fois.

Exemple: somme de 1+2+3

`Dim s, i As Integer`
`s = 0`
`i = 1`
`Do While (i <= 3)`
`  s += i`
`  i += 1`
`Loop`
 	
`Console.WriteLine(s)`
Exécution dans les 3 cas:
6
	`Dim s, i As Integer`
	`s = 0`
	`i = 1`
	`While (i <= 3)`
	`  s += i`
	`  i += 1`
	`End While`
	
	`Console.WriteLine(s)`
		`Dim s, i As Integer`
		`s = 0`
		`i = 1`
		`Do`
		`  s += i`
		`  i += 1`
		`Loop While (i <= 3)`
		
		`Console.WriteLine(s)`

Pour sortir d'une boucle do ... loop, on utilise Exit Do.

Exercice : Donner un programme permettant de chercher une valeur dans un tableau, en utilisant Exit Do.

Solution:

```
Dim tab() As Integer = {3, 4, 7}
Dim val As Integer = 7
Dim i As Integer = 0
Dim trouve As Boolean = False

Do While i < tab.Length
    If val = tab(i) Then
        trouve = True
        Exit Do
    End If
    i = i + 1
Loop

If (trouve) Then
    Console.WriteLine(val & " trouvé")
```

```
        Else
            Console.WriteLine(val & " non trouvé")
        End If
```

5.2.3. Boucle Répéter … jusqu'à

On sort de la boucle quand la condition est vérifiée.

Do instructions ***Loop Until*** (condition)	***Do Until*** (condition) instructions Loop
Test de la condition à la fin de la boucle → Exécution au moins une fois	Test de la condition au début de la boucle → Les instructions peuvent ne pas s'exécuter

Exemple: somme de 1+2+3

```
Dim s, i As Integer
s = 0
i = 1
Do Until (i > 3)
    s += i
    i += 1
Loop
Console.WriteLine(s)
```

Exécution dans les 2 cas:
6

```
Dim s, i As Integer
s = 0
i = 1
Do
    s += i
    i += 1
Loop Until (i > 3)
Console.WriteLine(s)
```

De même, ***exit do*** fait sortir d'une boucle do ... loop

Chapitre 2: Introduction à la Programmation événementielle en VB.NET

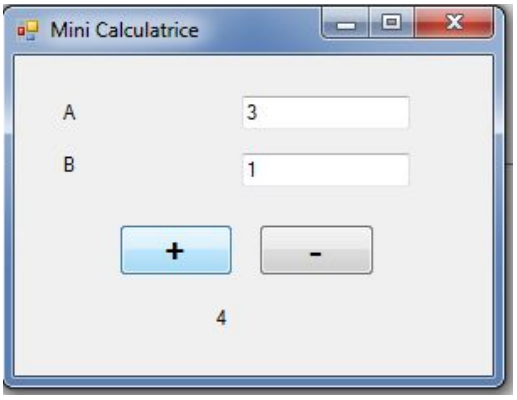
1. Principe de la Programmation événementielle

A part la programmation séquentielle, où les instructions s'exécutent de manière séquentielle, on peut utiliser VB.NET pour réaliser de la programmation par événements, c'est à dire programmer des procédures qui s'exécutent quand un *événement* est déclenché. La plupart du temps, l'événement est déclenché par l'utilisateur du programme.

Nous nous intéressons aux applications avec multifenêtres (Windows Applications). Ces applications sont constituées de deux parties principales:

1) Partie visuelle: des fenêtres, des boutons, des labels. C'est ce qui est vu par l'utilisateur final de l'application.
2) Code VB.NET "derrière" la partie visuelle. C'est au développeur de réaliser cette partie.

Exemple : Mini Calculatrice



L'utilisateur de l'application va interagir avec la fenêtre de la mini-calculatrice.

- L'utilisateur saisi deux entiers A et B
- Il clique sur un bouton d'addition ou de soustraction
- Le système affiche le résultat.

Quand on clique sur le bouton de la somme, on voit apparaitre le résultat. → Le clique sur le bouton est un **<u>événement</u>** qui a causé l'affichage de la somme.

Quand un événement a lieu, une procédure liée à cet événement est appelée. Le développeur de l'application doit développer le code VB.NET relatif à ces procédures. Une procédure événementielle est donc une procédure classique mais qui s'exécute quand un événement particulier se produit. (Il est à noter qu'elle peut être aussi appelée dans du code comme une procédure classique). Dans notre exemple, on doit par exemple développer la procédure au clique sur le bouton de l'addition. Cette procédure va :

1) Récupérer l'entier A saisi par l'utilisateur
2) Récupérer l'entier B saisi par l'utilisateur
3) Faire la somme
4) Afficher le résultat.

En pratique, la procédure événementielle générée pour un objet suit la syntaxe suivante (à ne pas modifier ces paramètres):

```
Private Sub NomObjet_NomEvénement ( paramètres )
Instructions
End Sub
```

2. Généralités sur les contrôles

2.1. Utilité des contrôles basiques

La partie visuelle du projet est composée principalement d'un ou de plusieurs formulaires (Forms). Un formulaire représente une fenêtre (Window). Dans ce formulaire, on va ajouter les éléments graphiques dont on a besoin.

Nom de l'élément	Utilité
Form (feuille)	C'est le conteneur graphique des contrôles de l'application
Label (étiquette)	Afficher un texte statique : un libellé.
Text Box (zone de texte)	rentrer une valeur saisie, on peut aussi l'utiliser pour afficher du texte mais ce n'est pas recommandé. (utiliser plutôt le label pour l'affichage)

Button (bouton de commande)	Lancer l'exécution d'une procédure événementielle (généralement suite au click sur ce bouton)
ListBox	Afficher une liste de valeurs.
ComboBox	Combiner l'utilité des contrôles ListBox et TextBox
PictureBox	Afficher une image dans un cadre. (Il peut être redimensionné en fonction de l'image en utilisant Autosize = True)
RadioButton (bouton radio)	Sélectionner **une seule** option parmi plusieurs.
Check Box (case à cocher)	choisir **une ou plusieurs** options parmi d'autes.
GroupBox	Regroupe un ensemble de contrôles graphique dans un seul cadre au sein du formulaire.

2.2. Propriétés

VB.NET est un langage orienté objet. Les éléments graphiques (qu'on appelle des **contrôles**), sont des objets de classes:

Par exemple: un bouton nommé "bouton1" est une instance de la Classe "`System.Windows.Forms.Button`". Un bouton a donc des propriétés (des caractéristiques, comme les attributs des objets vus en Java) et des méthodes prédéfinies qu'on peut utiliser.

Pour chaque élément, on doit changer les propriétés des éléments. Il s'agit généralement de changer 2 propriétés principales: "nom" et "texte". Ceci se fait à partir de la fenêtre "propriétés".

Le nom : sert à désigner l'élément graphique dans le code.
Le texte : c'est le texte à afficher dans l'interface.

Chaque élément graphique a des propriétés communes avec les autres éléments, et d'autres qui lui sont spécifiques. Voici quelques propriétés communes couramment utilisées:

- Name : nom interne de l'objet utilisé dans le code
- Text :Texte affiché à l'écran (sauf pour les champs de saisie)
- BackColor : couleur du fond
- Enabled : Contrôle activé ou non
- Font : police de caractères
- ForeColor : couleur de l'écriture
- Visible : Contrôle visible ou non

- Width : hauteur de l'objet
- Height : largeur de l'objet

Pour accéder à une propriété à partir du code VB, on utilise la syntaxe suivante:

NomObjet.Propriété

Exemples:

txtNom.Text="moi" → modifier le texte de txtNom

cmdQuitter.enabled =false → rendre le bouton désactivé.

De même, pour accéder à une méthode propriété à partir du code VB, on utilise la syntaxe suivante:

NomObjet.Méthode ()

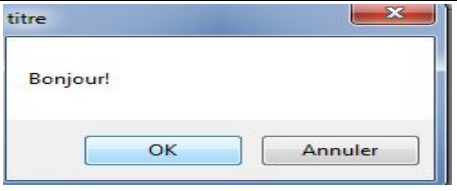
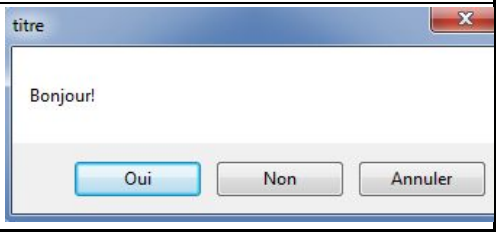
Exemples :

frmPrincipale.hide () → cacher la fenêtre, elle n'est plus visible

frmPrincipale.show () → rendre la fenêtre visible

2.3. Instructions particulières

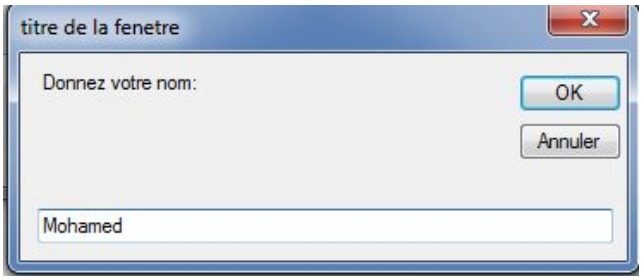
L'instruction MsgBox("Bonjour ISET Jendouba!") permet d'afficher une fenêtre simple contenant le texte "Bonjour! " avec un bouton OK. Ceci peut se faire aussi en utilisant l'instruction suivante: MessageBox.Show("Bonjour ISET Jendouba!") Ceci sert à simplifier au développeur la création des fenêtres simples couramment utilisées surtout pour informer l'utilisateur. On peut ajouter d'autres paramètres en fonction du besoin. Ci-dessous quelques cas d'utilisation:

Instruction	Résultat
MessageBox.Show("bonjour!", "titre")	
`MessageBox.Show("Bonjour! ", "titre", MessageBoxButtons.OKCancel)`	
MessageBox.Show("Bonjour!", "titre", MessageBoxButtons.YesNoCancel)	

L'instruction :

```
Dim reponse As String = InputBox("Donnez votre nom:", "titre de la fenetre")
```

Permet d'afficher une fenêtre pour récupérer une réponse de la part de l'utilisateur. La réponse est stockée dans la variable "reponse" de type String.

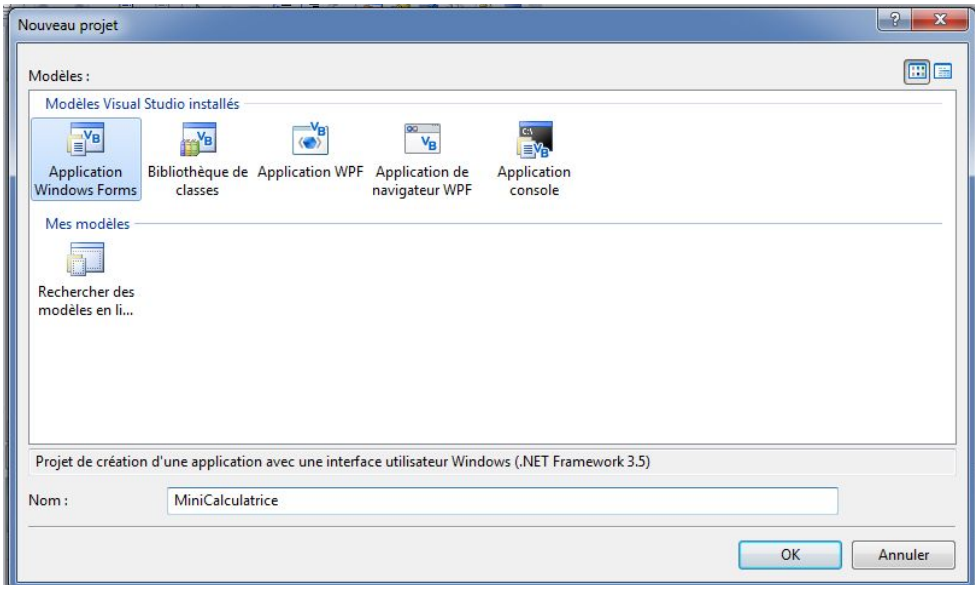


3. Exemple Pratique: Mini Calculatrice

3.1. L'environnement de développement de VB.NET

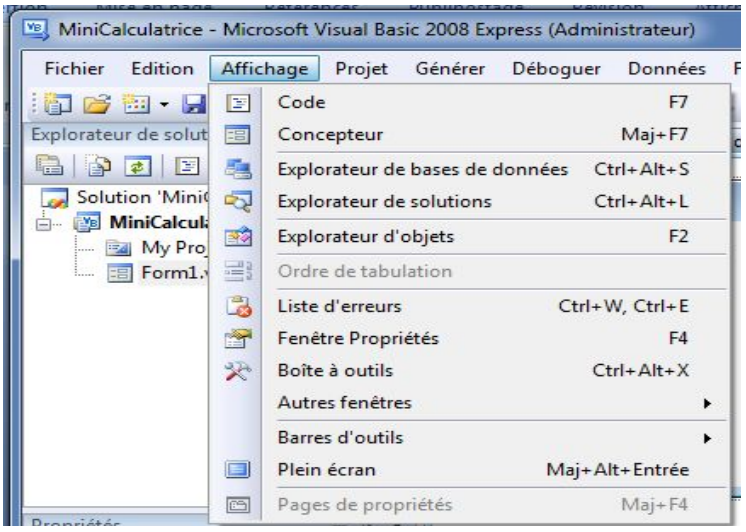
Nous allons opter pour l'utilisation de l'environnement gratuit: Microsoft Visual Basic 2008 Express. (On peut aussi utiliser Microsoft Visual Studio).

On commence par créer un projet: fichier>nouveau Projet puis Choisir "ApplicationWIndows Forms" et choisir un nom pour le projet.



Remarques :

- SI une des ces fenetres n'est pas affichée, on peut la faire apparaitre en utilisant le menu AFFICHAGE dans la barre des menus :



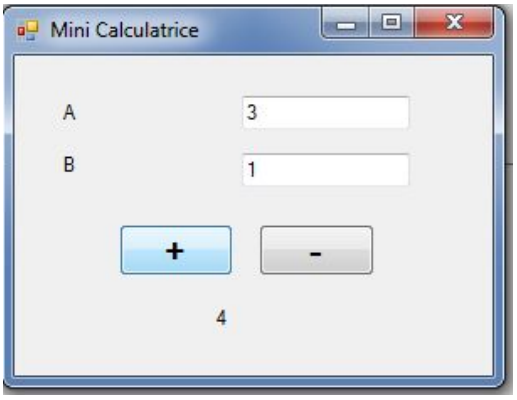
- On peut changer l'emplacement de ces fenêtres en les glissant pas la souris.
- L'onglet de la boite à outils contient plusieurs types d'éléments, qui sont organisés par type.

Pour créer nos applications, on va suivre une démarche contenant ces trois étapes:

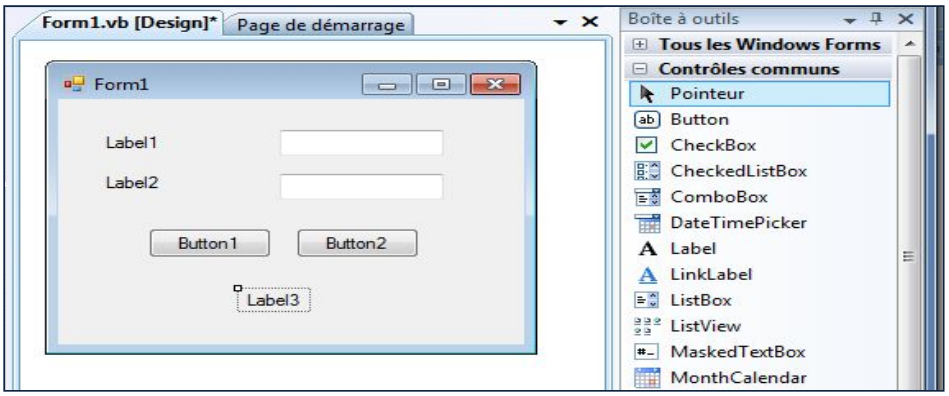
1) Préparer l'interface graphique
2) Déterminer les événements
3) Développer la procédure liée à chaque événement.

3.2. Préparer l'interface graphique

Il s'agit de concevoir l'allure de l'interface graphique de l'application, et choisir les différents éléments qui la composent. Dans notre cas, on va adopter pour cette interface:



À partir de la boite à outils, on choisi le groupe "Contrôles communs" et on glisse 3labels, 2 textBox et 2 boutons pour avoir l'allure suivante :



Pour chaque élément, on change 2 propriétés principales: "nom" et "texte". Ce ci se fait à partir de la fenêtre "propriétés".

ément	opriété	leure
bel1	**om**	**A**
	xte	
bel2	**om**	**B**
	xte	
bel3	**om**	**Resultat**
	xte	
utton1	**om**	**nAddition**
	xte	
utton2	**om**	**nSoustraction**
	xte	
xtBox1	**om**	**A**
	xte	
xtBox2	**om**	**B**
	xte	

3.3. Préciser les événements

Dans notre mini-calculatrice, nous avons deux événements:

a) Clique sur le bouton de l'addition
b) Clique sur le bouton de la soustraction

3.4. Développer les procédures liées aux événements.

Une fois les événements sont précisés, on passe à développer le code relatif aux procédures liées à chaque événement. En pratique, on double clique sur le

bouton de l'addition, on voit afficher la fenêtre du code associé au formulaire, elle contient les lignes suivantes:

```
    Public Class Form1
       Private Sub btnSomme_Click(ByVal sender As System.Object,
     ByVal e As System.EventArgs) Handles btnSomme.Click
    ……..
        End Sub
d Class
```

VB.Net est un langage orienté objets, entre autres les formulaires sont des classes. Dans notre cas, on a la classe Form1 :

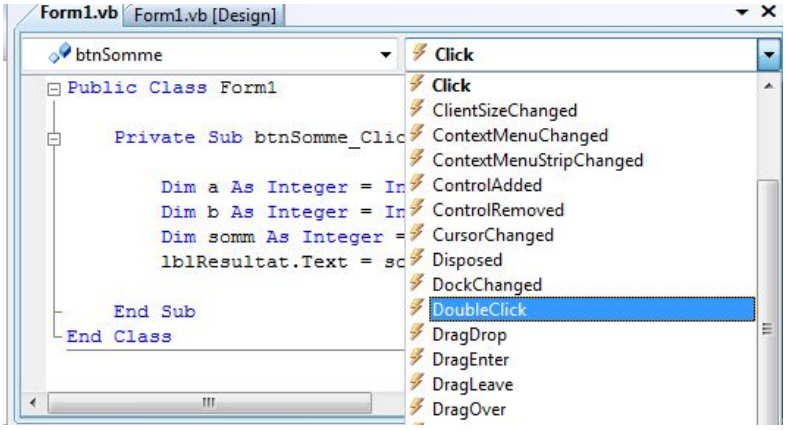
```
    Public Class Form1
            .........
    End Class
```

Dans cette classe, on trouve la procédure liée au clique sur le bouton btnSomme, elle est générée automatiquement par l'environnement de développement que nous utilisons.

```
     Private Sub btnSomme_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles btnSomme.Click
          . . . . .
    End Sub
```

C'est Cette Procédure qui va être appelée quant l'événement "clique sur le bouton btnSomme" a lieu → On parle alors d'une procédure liée à un événement. Ceci est la base de la programmation événementielle.

Il est à noter qu'on peut ajouter d'autres événements sur le même élément graphique : double clique… comme indiqué dans la figure qui suit :



Dans notre cas, on ajoute le code suivant:

```
        Dim a As Integer = Integer.Parse(txtA.Text)
        Dim b As Integer = Integer.Parse(txtB.Text)
        Dim somm As Integer = a + b
        lblResultat.Text = somm
```

Exercices

1) Donner le code correspondant au bouton de la soustraction
2) Ajouter deux boutons: multiplication et division.

4. Manipulation de quelques éléments

4.1. Les formulaires

Un formulaire représente une fenêtre (Window). C'est le conteneur principal des autres éléments graphiques. Il s'git de la classe System.Windows.Forms.Form de la plateforme .NET.

La classe Form dispose évidemment des propriétés communes déjà vues, particulièrement, la propriété "text" permet de manipuler le titre de la fenêtre (placé dans la barre de titre). Deux méthodes couramment utilisés sont :

la méthode Show() qui permet d'ouvrir une fenêtre à partir d'une autre
la méthode Close() qui permet de fermer une fenêtr
la méthode Hide() qui permet de rendre la fenêtre invisible.

Pour identifier le formulaire courant, on peut utiliser le mot clé "Me" (semblable à la référence "this" en Java).

Exemple: On se propose de modifier le titre, la taille est la couleur de l'arrière plan lors du chargement de la fenêtre. → On va donc développer la procédure liée à l'évènement *Form Load*.

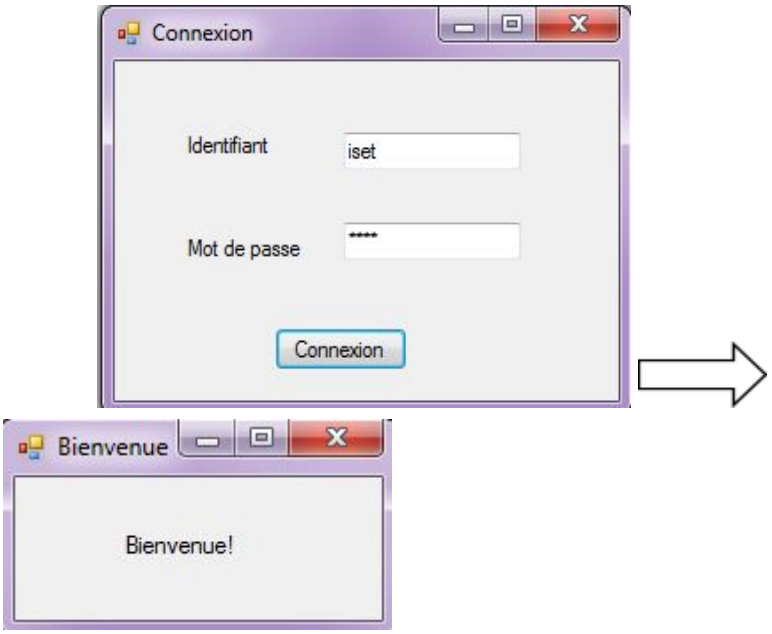

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Me.Text = "C'est le titre "
        Me.Size = New System.Drawing.Size(250, 90)
        Me.BackColor = Color.Blue

    End Sub
End Class
```

Exercice:

Une fenêtre d'authentification contient un champ de saisie pour l'identifiant et un autre pour le mot de passe de l'utilisateur. Si les deux sont égales à la chaine "iset", alors on ouvre une autre fenêtre (la fenêtre principale de l'application). Donner le code de la procédure événementielle liée au click sur le bouton connexion.



Solution:

```
Public Class frmAuthentification

    Private Sub cmdConnexion_Click(…)
        If txtId.Text = "iset" And txtPassword.Text = "iset" Then
             Me.Hide() 'Fermer la fenêtre actuelle
              frmBienvenue.Show() 'ouvrir la nouvelle fenetre
        Else
            MessageBox.Show("Login ou mot de passe incorrect")
        End If
    End Sub
End Class
```

4.2. Les étiquettes et les zones de texte

Une étiquette sert à afficher du texte sur la fenêtre alors qu'une zone de texte sert à récupérer le texte saisi par l'utilisateur. La première est une instance de la classe *System.Windows.Forms.Label* de la plateforme .NET alors que la deuxième représente une instance de la classe *System.Windows.Forms.TextBox*. Puisqu'ils servent principalement à afficher ou saisir du texte, la propriété généralement utilisée pour ces deux contrôles est la propriété *Text*. On peut éventuellement changer la police et la couleur des caractères.

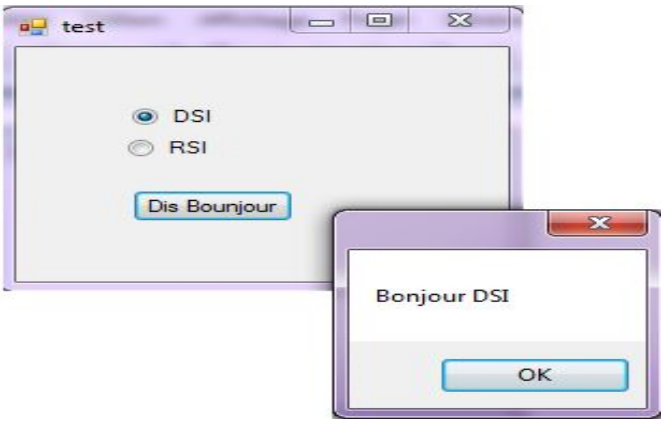
4.3. CheckBox et radioButton

La propriété qui stock l'état de sélection d'un checkBox ou un RadioButton est la propriété **`Checked`**, de type *Booléen.* La valeur *True* veut dire que le contrôle est choisie. Il

est à noter bien que VB se charge de mettre à jour la propriété *checked* une fois que la sélection est modifiée.

L'événement lié à la modification de l'état de la propriété checked est : "`CheckedChanged`". Par exemple: " `CheckBox1_ CheckedChanged` " est le nom de la procédure événementielle appelée quand la valeur de selection de l'objet a été changée.

Exercice: Donnez le code de la procédure événementielle liée au clique sur le bouton qui permet d'afficher: Bonjour+ DSI ou RSI selon le bouton Radio sélectionné.



Solution:

```
Private Sub cmdBonjour_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdBonjour.Click
        If optDSI.Checked Then
            MessageBox.Show("Bonjour DSI")
        Else
            MessageBox.Show("Bonjour RSI")
        End If

    End Sub
```

4.4. Le contrôle group Box

Un *GroupBox* est une fenêtre qui peut être placée sur un formulaire pour regrouper des contrôles. Tous les contrôles (qui peuvent être différents: labels, boutons…)placés sur le GroupBox appartiennent au même groupe. La propriété "Text" permet de donner un titre au GroupBox.

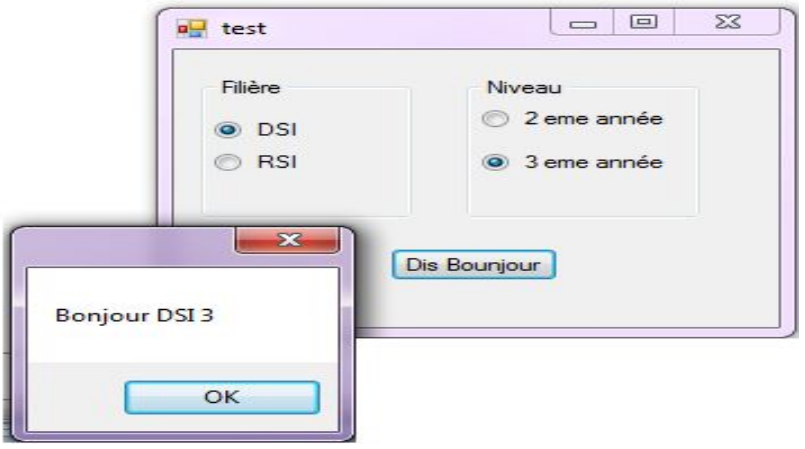
On a vu que si plusieurs RadioButton sont placés dans un formulaire, une seule option peut être choisie. Cependant, si on a besoin d'utiliser deux groupes de choix dans lesquels on peut sélectionner deux choix non exclusifs: un dans le premier groupe et un dans le second, ceci n'est pas possible s'ils se trouvent dans la même fenêtre. Pour résoudre ce problème, il suffit de:

- placer chaque groupe d'options dans un GroupBox à part

- ou bien placer un groupe d'options sur le formulaire et l'autre dans un groupe box.

Exercice :

Ajouter à l'interface de l'exercice précédent deux boutons radio pour le niveau: 2ème année et 3ème année.



Correction:

```
Public Class test

    Private Sub test_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        optDSI.Checked = True
        opt2.Checked = True

    End Sub


    Private Sub cmdBonjour_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdBonjour.Click
        Dim texte As String = "Bonjour"

        If optDSI.Checked Then

            texte += " DSI"
        Else
            texte += " RSI"
        End If
        If opt2.Checked Then

            texte += " 2"
        Else
            texte += " 3"
        End If
        MessageBox.Show(texte)
    End Sub
```

```
End Class
```

4.5. Les listes

Les des type de listes que nous avons vues sont : listBox et comboBox. Dans les deux cas, la liste des valeurs est stockée dans la propriété Items.

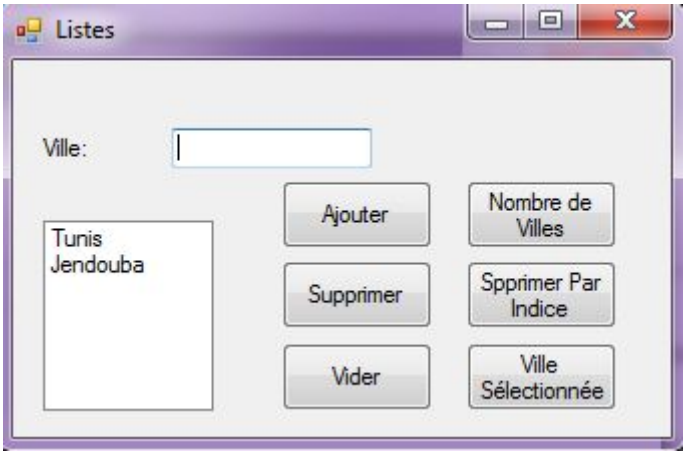
On peut initialiser cette liste lors de la conception à partir de la fenêtre *Properties de l'IDE (une valeur par ligne).* Cette liste peut aussi être mise à jour (ajout, suppression) de manière dynamique à partir du code VB.NET. Ci-dessous quelques méthodes et propriétés utiles pour le cas de la listBox.

NomListBox. Items .Add(valeur)	ajoute la valeur à la liste
NomListBox. Items .Remove (valeur)	enlève l'élément *valeur* de la *liste*
NomListBox. Items .RemoveAt (i)	enlève l'élément d'indice *i* de *la liste (i commence à partir de 0)*
NomListBox. Items. Clear	enlève tous les éléments de *List*
ListBox.Items.Count	donne le nombre d'éléments dans *List*
NomListBox.items (i)	retourne l'item d'indice *i* de la *liste (i commence à partir de 0)*
NomListBox. SelectedItem	retourne l'élément sélectionné
NomListBox. Sorted	Si elle est égale à *True alors la liste sera* maintenue triée par ordre alphabétique croissant.

Pour récupérer la valeur saisie dans le comboBox, on écrit: nomComboBox.text .

Exercice:

Soit l'interface suivante:



Question: Donnez le code VB.NET relatif au différents boutons.

Solution:

```
Public Class listes

       Private Sub cmdAjouter_Click(…)
        lstVilles.Items.Add(txtVille.Text)
        txtVille.Text = ""
    End Sub

    Private Sub cmdSupprimer_Click(…)
        lstVilles.Items.Remove(txtVille.Text)
        txtVille.Text = ""
    End Sub


    Private Sub cmdSupprIndice_Click(…)
        lstVilles.Items.RemoveAt(Integer.Parse(txtVille.Text))
    End Sub

    Private Sub cmdVider_Click(…)
        lstVilles.Items.Clear()
    End Sub


    Private Sub cmdSelect_Click(…)
        MessageBox.Show("Vous avez selectionné" + lstVilles.SelectedItem)
    End Sub

    Private Sub cmdTaille_Click(…)
        MessageBox.Show("Il existe " + lstVilles.Items.Count.ToString + "
Villes")
    End Sub

End Class
```

Bibliographie

O. Moursli et N. Souchon, VISUAL BASIC .NET : TUTORIAL , UNIVERSITE CATHOLIQUE DE L'OUVAIN, Institut d'Administration et de Gestion

Serge Tahé, APPRENTISSAGE DU LANGAGE VB.NET, ISTIA, Université d'Angers, Mars 2004

Site MSDN, http://msdn.microsoft.com/fr-fr/library/bb727317.aspx , http://msdn.microsoft.com/fr-fr/library/47zceaw7(v=vs.90).aspx , 2012

Mohamed Salah Bou Hlel, cours POO, ISETJ, Tunisie ,2010

Heithem Mezni, cours programmation évènementielle en VB.NET, ISETJ , Tunisie, 2011