1.4. LANGAGE DE PROGRAMMATION ORIENTE OBJET VISUAL BASIC.NET

Visual Basic.Net est un langage qui travaille dans le cadre d'un environnement *NET* commun. Cela apporte des changements considérables dans la technologie de conception, de programmation et de compilation des applications. Il faut savoir :

1. Le texte de programmation élaboré de *VB.NET* est compilé jusqu'un code intermédiaire *MSIL* (*Microsoft Intermediate Language*), représentant un ensemble d'instructions indépendant du processeur qui définissent le chargement, l'initialisation, le stockage, l'appel, l'exécution de calculs, gestion de l'exécution elle-même, accès à la mémoire, traitement des exceptions etc.
2. Démarrage du compilateur *JIT* (*Just-In-Time*) qui est élaboré exprès pour la plate-forme utilisée. *CLR* fournit des compilateurs *JIT* pour chaque langage inclus dans l'environnement *NET.* Chaque compilateur *JIT* fonctionne d'après le schéma suivant : lors de l'appel d'une méthode le compilateur convertit *MSIL* jusqu'un code exécutable optimum pour la plate-forme, modifie l'enveloppe pour une exécution directe suivant l'emplacement et la plate-forme. Cela accélère les appels ultérieurs de la méthode.
3. Pour atteindre une certaine indépendance on utilise des données descriptives (*métadonnées*). Elles sont sauvegardées dans un fichier appelé *manifeste*. Les métadonnées sont une conditions nécessaire pour l'environnement *NET* car elles signalent des types exportés ou référés, décrivent des classes de bases, des interfaces implémentées et la visibilité de chaque type. Les métadonnées sont enveloppées dans un fichier transférable *PE* (*Portable Executable*) .
4. Lors de l'exécution du code de gestion, *CLR* fournit un nombre de services liés à la gestion de la mémoire, l'entretien d'un niveau de sécurité, l'interaction avec un code *non gérable* (créé en dehors de l'environnement *NET)*, entretien d'un traçage intralingue, entretien de versions différentes etc.
5. Dans l'environnement *NET* on entend sous la notion de *module* un fichier exécutable séparé ou toute une bibliothèque. *L'assembleur* représente un regroupement logique d'une multitude de modules nécessaires pour l'exécution de l'application élaborée. Il contient un *manifeste* contenant des métadonnées qui représentent une liste des fichiers inclus et les moyens de leur localisation. Lors du chargement d'un programme on localise son assembleur par des méthodes d'essais heuristiques (*probing*) et on crée une référence auprès de l'assembleur.

Visual Basic.Net est un des langages de programmation orienté objet inclus et entretenus par l'environnement *NET.* Donc les règles et les caractéristiques communes du langage sont *en correspondance* avec les autres langages. Voilà pourquoi il existe de différences importantes avec les versions antérieures de ce langage.

Les propriétés communes caractéristiques pour les langages orientés objet sont également valable pour *Visual Basic. Net*. Elles sont comme suit :

L'objet est un élément de base des langages orientés objet. Lors de l'exécution l'objet est identifié de façon univoque. Celui-ci possède de propriétés spécifiques, entretient un ensemble d'événements et fonctionnalité prédéfinie dans les méthodes qu'il exécute.

La classe définit les propriétés, les méthodes et les événements communs caractérisant un groupe d'objets.

L'abstraction représente une possibilité de séparer l'essentiel de ce qui ne l'est pas dans les objets. Lors d'une conception concrète ou d'une exécution, une partie de la fonctionnalité, prédéfinie dans les objets pourrait devenir inutile (en plus). Grâce à l'abstraction cette fonctionnalité est ignorée

Capsulage (*encapsulation*) est une méthode d'application de l'abstraction. Par le capsulage on cache les détails de l'objet qui ne sont pas essentiels pour le moment. Par exemple cacher la réalisation interne de l'objet.

L'héritage est la possibilité d'obtenir une *sous-classe* à partir d'une *classe de base*. La sous-classe hérite les propriétés, les événements et les méthodes de la classe de base. Certaines méthodes de ma classe de base peuvent être élargies dans la sous-classe. L'héritage permet la projection et la conception d'applications assez compliquées à partir des classes de base.

Le polymorphisme est la possibilité rend possible l'existence d'un objet sous différentes formes. Par exemple à partir d'une classe de base on a créé plusieurs sous-classes. Elles héritent les méthodes de la classe de base. Il est possible qu'une même méthode soit différemment modifiée dans chaque sous-classe. Donc, bien que la méthode existe sous le même nom dans les sous-classes elle possède une fonctionnalité différente.

1.4.1. Données types et variables

La maîtrise d'un langage de programmation concret est liée à l'étude de ses moyens d'expression, la sémantique et la syntaxe des opérateurs, des possibilité de construite un texte de programmation, de conception et d'entretien d'une fonctionnalité définie de l'élaboration d'un système d'interfaces etc.

VB.NET comme un des langages de l'environnement *NET* entretient les données types suivantes :

Byte – sauvegarde des données binaires

Integer – un entier sauvegardé en 4 octets
Long – un entier sauvegardé en 8 octets
Short – un entier sauvegardé en 2 octets

Single – nombre à virgule flottante (précision unique), nombre à virgule sauvegardé en 4 octets
Double – flottante (précision double), sauvegardé en 8 octets
Décimal – nombres sauvegardés en 16 octets

Boolean – sauvegarde en 2 octets et ayant des valeurs *True* et *False*
Char – sauvegarde de symboles séparés en 2 octets

String – pour la sauvegarde de lettres et de chiffres

DateTime – chiffre contenant une date et l'heure suivant le standard de *IEEE*, sauvegardé en 8 octets

Object – objet spécial pour la sauvegarde de tout type de données.

La variable représente un identificateur unique valable pour un certain domaine d'action, comme par exemple une procédure, un formulaire, un module etc. L'identificateur de la variable représente une succession de lettres et de chiffres ne dépassant pas 255, commençant par une lettre. La variable sauvegarde un type de données bien défini.

1.4.2. Déclaration et initialisation de variables

La déclaration de la variable et de son type pourrait se faire de façon explicite, par défaut et implicite, par exemple :

Dim A As Integer déclaration explicite du type entier
Dim B% déclaration du type entier par défaut

Dans ce cas on utilise des caractères spéciaux mis à la fin de l'identificateur de la variable et indiquant son type :
% - *Integer*, **&** - *Long*, **!** – *Single*, **#** - *Double*, **@** - *Decimal* и **$** - *String*

C = 6 – implicite, lors d'une première rencontre d'affectation.

La déclaration implicite est héritée par les anciennes versions du langage et est admise uniquement lorsqu'un opérateur permettant une déclaration implicite est prévu.

Option Explicit Off – déclaration implicite permise
On – interdite, par défaut

Lors de l'exécution du programme, les variables sont créées au moment de leur utilisation et non de leur déclaration. Pour la création immédiate d'une variable il est nécessaire d'inclure dans l'opérateur le mot clé *New*, par exemple

Dim A *As Integer* ou *Dim* A *As Integer = New Integer()*
A = *New Integer()* ou *Dim* A *As New Integer()*

Lors de la déclaration les variables acquièrent une valeur par défaut : *0* – pour les numéraux et *False* pour les variables type *Boolean* .
La valeur initiale d'une variable pourrait être prédéfinie explicitement lors de sa déclaration ou par l'intermédiaire de l'affectation.

Dim A *As Integer* ou *Dim* A *As Integer = 3*
A = 3

Lors du processus de l'exécution du programme les variables peuvent obtenir des valeurs différentes. Le retour à une valeur par défaut, valable pour le type de variables se fait par le mot clé *Nothing,* par exemple

Dim Z As Single = 3.0
Z = Nothing - obtient une valeur 0 (zéro) par défaut.

1.4.3. Tableaus, méthodes et collections

Le tableau représente un ensemble de variables du même nom, stockant des données du même type et identifiées selon leur emplacement dans le tableau par des index ou un système d'index (32 index maximum). La déclaration des tableaus se fait par l'opérateur *Dim.*

Dim XX (100) As Single déclare un tableau de 101 éléments
avec des index de 0 à 100
Dim XX () As Single = New Single (100) ()

Lors de l'exécution du programme un changement des limites supérieures des index est possible, mais sans changement de leur nombre. Par exemple :

Dim Z() As Integer déclare un tableau unidimensionnel,
ReDim Z(50) définit la valeur courante
de la limite supérieure de l'index.

Dim K (3,4) As Double
ReDim K (5,6) change les limites supérieures.

Chaque changement des valeurs limites du tableau entraîne une perte des valeurs déjà saisies de ses éléments. Le mot clé *Preserve* permet la sauvegarde des valeurs.

Dim Z(50) As Integer
ReDim Preserve Z(100) As Integer les premiers 50 éléments ne changent pas de contenu.

Les tableaux bi- et pluridimensionnels permettent un changement de la limite du *dernier* index. Le cas contraire une erreur est indiquée.

Lorsqu'un tableau n'est plus nécessaire dans le processus du traitement il est souhaitable que la mémoire qu'il a réservée soit libérée. L'opérateur *Erase* le permet, par exemple :

Erase Z, K libère la mémoire utilisée par les deux tableaus.

Chaque tableau réservé représente un objet de la classe *System.Array*. Donc les tableaus représentés comme des objets possèdent des méthodes dont les plus utilisées sont :

Dim P(30,50,60) As Single
*P.GetUpperBound(***0***)* retour de la limite supérieure du premier index.
*P.GetLowerBound (***1***)* retour de la limite inférieure du deuxième index.
*P.GetLength(***2***)* retour du nombre des éléments (61) du troisième index.
*P.SaveValue(***5.5***,1,20,30)* enregistre 5.5 dans un élément du tableau défini par une combinaison indexée 1,20,30.

VB.NET rend possible la création d'un tableau de tableaus. Les éléments de ce tableau peuvent être de types différents. Le tableau doit être obligatoirement du type *Object.*

Dim FirstName(20) As String
Dim BDay(30) As DateTime
Dim PYear (25) As Integer

Dim Marray(3) As Object
Marray(1) = FirsName
Marray(2) = BDay
Marray(3) = PYear

Les tableaus sont faciles à utiliser lorsque l'information stockée est du même type et lorsque le nombre de leur éléments reste le même. L'insertion d'un élément entre d'autres éléments du tableau se fait uniquement par voie de programmation. Il est nécessaire de concevoir une fonctionnalité exécutant insertion, suppression ou changement des positions des éléments du tableau.

VB.NET permet la création de collections, qui sont beaucoup plus souples en ce qui concerne l'ajout et la suppression d'éléments dans la collection. Pour s'y faire il faut :

1.4.4. Création d'une collection d'éléments utilisateurs.

Dim MyCollection As New Collection () création d'une collection appelée *MyCollection* de la classe des collections *Collection.*

Chaque élément de la collection possède son propre identificateur. Cela peut être un *index* qui commence à partir de 1 pour les collections créées et à partir de 0 pour le reste. *Le string* peut être également utilisé en sa qualité d'identificateur.

1.4.5. Ajout d'un élément à la collection à l'aide de la méthode A*dd.*

MyCollection.Add ("Ivan*")*
ajoute un nouvel élément. L'index est obtenu automatiquerment.

MyCollection.Add("Ivan*","*46*")*
ajoute un nouvel élément avec un identificateur 46.

MyCollection.Add ("Dimitar*","*44*","*46*")*
ajoute un nouvel élément avec un identificateur 44, placéavant l'élément avec un identificateur 46.

MyCollection.Add ("Stephan*","*45*" , ,"*44*")*
ajoute un nouvel élément avec un identificateur 45, placé après l'élément avec un identificateur 44.

1.4.6. Suppression d'éléments de la collection.

On utilise la méthode *Remove*.

*MyCollection.Remove(***1***)*	supprime le premier élément
*MyCollection.Remove(***"44"***)*	supprime l'élément avec un identificateur 44.

1.4.7. Extraction d'un élément de la collection.*

On utilise la méthode *Item*.

MyCollection.Item(1)	extrait le premier élément de la collection
MyCollection.Item("46")	extrait l'élément avec un identificateur 46

1.4.8. Dénombrement des éléments de la collection.

```
Dim N As Integer
N = MyCollection.Count()
```

1.4.9. Opérateurs de Visual Basic.Net.

Les opérateurs dans *VB.NET* peuvent être groupés suivant différents indices. Par exemple :

- pour la déclaration de variables, d'objets et pour la création d'exemplaires de classe etc.
- pour l'exécution de calculs arithmétiques ou logiques.
- pour la vérification des conditions et l'organisation de cycles.
- pour le travail avec des fichiers et des bases de données etc.

If et *Select Case* font partie du groupe des opérateurs de gestion:

If condition *Then* opérateur
If condition *Then* opérateur 1 : opérateur 2 : opérateur 3
If condition *Then* opérateur 1 *Else* opérateur 2

```
If condition Then
    Groupe d'opérateurs
End If
```

If condition *Then* *If* condition 1 *Then*

```
    Groupe d'opérateurs 1              groupe d'opérateurs 1
Else                                 Else If condition 2 Then
    Groupe d'opérateurs 2              groupe d'opérateurs 2
End If                                End If
```

```
If condition 1 Then
    Groupe d'opérateurs 1
Else If condition 2 Then
    Groupe d'opérateurs 2
Else
     Groupe d'opérateurs 3
End If
```

L'opérateur *Select Case* peut suivre l'exemple suivant:

```
Select Case condition
Case 1,3,5,7        'liste de valeurs     Groupe d'opérateurs 1
Case 10 to 100                        gamme de changement
        Groupe d'opérateurs 2
Case Is > 200                          ' plus grand que ....
         Groupe d'opérateurs 3
Case Else                               ' tout le reste
        Groupe d'opérateurs 4
End Select
```

Les opérateurs d'organisation de cycles sont comme suit :

```
Dim I As Integer                    Dim I As Integer
For I = 1 To 100 Step 2             For I = 100 To 1 Step -2
  Groupe d'opérateurs                Groupe d'opérateurs
     [Exit For]                          [Exit For]
Next I                              Next I
```

```
Dim Mac() As String = ("Ivan", "Dimitar","Stephan")
Dim Imac As String
For Each Imac  In Mac
          Groupe d'opérateurs
          [Exit For]
Next
```

```
While condition
          Groupe d'opérateurs
```

[*Exit While*]
End While

Do while condition
Groupe d'opérateurs
[*Exit Do*]
Loop

Do Until condition
Groupe d'opérateurs
[*Exit Do*]
Loop

Les opérateurs structurés *If, Select Case, For, While, Do* While et *Do Until* peuvent être placés l'un dans l'autre sans se croiser.

1.4.10. Procédures dans *VB.NET*

La procédure représente un groupe d'opérateurs, portant un seul nom identificateur, possédant des arguments et appelé à l'exécution à l'aide du nom et des valeurs des arguments. De cette façon on atteint une grande souplesse du texte de programmation et une possibilité d'exécution de la procédure à partir de différents endroits de l'application et avec des données différentes.

Dans VB.NET les procédures sont du *type Sub, Function, Property* et *Event_Handing*. Chaque procédure à son propre modificateur d'accès à la procédure. Les modificateurs sont comme suit :

Public – la procédure est accessible à partir de n'importe quelle classe ou module de l'application.
Private – accessible uniquement à partir de la classe ou du module où elle est déclarée.
Protected – peut être appelée à l'exécution à partir de la classe ou du module où elle est déclarée, ainsi qu'à partir des classes dérivées de cette même classe.
Friend – accessible à partir de chaque classe ou module contenant sa déclaration ou à partir d'une classe dans le même environnement de noms.

1.4.11. Procédures du type *Sub*.

Ce sont des procédures qui possède une liste d'arguments. Le nom de la procédure ne retourne pas de résultat, mais sert uniquement à son identification en cas d'appel. Le nom de la procédure doit correspondre aux conditions dans lesquelles on définit le nom d'une variable.

Dans *VB.NET* le nom *Main* est réservée pour une procédure spéciale. Elle est exécutée première lors du démarrage du programme.

Vue générale de la procédure:

Modificateur *Sub* nom (liste d'arguments)
Groupe d'opérateurs
[*Exit Sub*] et/ou [*Return*] et/ou [*End*]
End Sub

La procédure est appelée à l'exécution par l'intermédiaire de l'opérateur *Call* ou par son nom. Au moment d'appel des arguments de la procédure on doit obtenir des valeurs concrètes. L'exécution commence par le premier opérateur et se termine au moment d'arrivée à un des opérateurs suivants *Exit Sub, Return, End* ou *End Sub.* Après l'exécution de la procédure la gestion est remise à l'opérateur qui suit celui qui l'a appelée.

1.4.12. Procédure du type *Function*

La procédure *Function* est homologue à *Sub* avedc la seule différence que le nom reçoit et retourne le résultat. L'appel de la fonction n'est pas indépendant mais se fait uniquement comme un élément d'opérateur. La vue générale de la procédure est la suivante :
Modificateur *Function* nom (liste d'arguments) *As* type du résultat
Groupe d'opérateurs
[*Exit Function*]
[*Return* résultat]
End Function

Les procédures *Property* et *Event_Handling* seront étudiées dans le chapitre concernant la création de classes et de contrôles.

1.4.13. Types d'arguments

Les arguments peuvent être transmis par valeurs et par adresses.
Dans le premier cas une copie de la variable originale est créée. Les changements de la variable, provoqués par l'appel de la procédure n'influencent pas la variable dans la partie appel. Dans la liste des arguments la transmission par valeurs est indiquée par

(*ByVal* nom de la variable *As* type de la variable)

Lorsque la variable qui représente l'argument en est une d'entrée de part sa valeur la remise de l'argument par valeur est recommandée. Le résultat ne retourne pas dans la procédure d'appel. Cela réduit la possibilité de diffusion d'une erreur de calcul apparue.

Dans le deuxième cas la procédure reçoit une référence vers l'emplacement de la variable originale. Donc la procédure peut changer sa valeur. On indique de la façon suivante :

(*ByRef* nom de la variable *As* type de la variable)

La remise par adresse permet à la procédure d'appel d'utiliser l'argument changé. De cette manière l'erreur apparue dans la procédure exécutée sera transmise par valeur à la procédure d'appel.

Les autres opérateurs et constructions de programme du langage de programmation orientée objet *VB.NET* seront étudies lors de la solution de certains problèmes définis.