2.2. VISUAL BASIC.NET – CLASSES ET OBJETS.

Il faut prendre en compte les définitions suivantes en travaillant avec des classes :
Module - unité de programmation indépendante, composante d'un projet où l'on peut définir des classes, des propriétés et des méthodes.
Structure – des types de données définis par l'utilisateur. On peut définir des variables et des méthodes.
Procédure – succession distincte d'opérateurs du langage orienté objet pour l'exécution d'une fonctionnalité prédéfinie.
Champs – variable déclarée dans une classe définie qui peut être utilisée par d'autres classes.
Membres – procédures, champs et méthodes définis dans la classe.
Interface – ensemble de définitions de propriétés et de méthodes que les classes peuvent réaliser. L'interface n'inclut pas des opérateurs exécutables. Il est recommandé d'utiliser des interfaces au cas où les classes se ressemblent peu. Le cas contraire il faut créer des classes dérivées.

Toutes les classes dans VB.NET sont des classes dérivées de la classe *Object*. Pour élaborer une classe il faut définir les composantes suivantes :

1. Modificateur d'accès à la classe élaborée. L'accès est réglementé par des mots clés.

Public – accès autorisé à partir du même projet, d'autres projets ou de blocs assemblés, créés du projet.
Private – accès du même module, classe ou structure,
Protected – accès autorisé dans la classe ou d'une classe dérivée,
Friend – accès autorisé uniquement dans le projet,
Protected Friend – accès autorisé dans le projet ou de classes dérivées.

2. Autorisation ou interdiction d'héritage de la classe par des classes dérivées.

Inherits – héritage de membres non personnels de la classe définie.
MustInherit – les classes dérivées peuvent hériter la classe de base,
NotInheritable – héritage interdit de la classe de base.

3. Le mot clé *Class* définit le début de la classe qu'on élabore. Toujours combinée avec *End Class*.
4. Nom propre unique de la classe. Les règles de nom de variable sont en vigueur.
5. Mot clé *Implements*. Usage obligatoire quand la classe réalise une interface.
6. Chaque interface élaborée est identifiée par un nom propre unique. La classe peut réaliser plus d'une interface.

2.2.1. Membres de la classe

Membres principaux de la classe:

- Constructeurs et Destructeurs,
- Méthodes
- Champs et propriétés.

Le constructeur sert à l'initialisation d'objets. Il n'est pas obligatoire. On considère de meilleur goût une classe possédant un constructeur. La procédure *Sub New* le permet. Cette procédure est exécutée lors de la création d'un objet de la classe. Son premier but est d'appeler le constructeur de la classe *Object*. C'est la classe principale, de base, dans VB.NET. Toutes les classes en dérivent. L'appel du constructeur se fait par le mot clé *MyBase.New()*. Dans la classe dérivée on appelle le code de la classe de base et les objets hérités sont initialisés. Après le constructeur on peut ajouter un code de programmation pour l'initialisation d'objets et de variables créés dans la classes dérivée.

La procédure *Sub New()* peut contenir des arguments entre parenthèses et déclarés, conformément à l'enregistrement adopté dans VB.NET. Lors de la création d'un objet on définit les valeurs concrètes des arguments. On peut l'utiliser pour le chargement initial des objets, suivant les arguments définis.

Le destructeur exécute la procédure *Sub Finalize*, prédéfinie dans la classe, représentant une méthode de sécurité de la classe *Object*. Le destructeur est destiné à libérer cette partie de la mémoire qui a été occupée par l'objet supprimé. Cette procédure n'est pas tout de suite appelé lors de l'exécution de l'application. Celle-ci est exécutée à l'initiative de l'environnement *NET* lors de la suppression d'un objet. Donc la suppression de l'objet est la libération ultérieure des ressources occupées peut se faire à des époques différentes. Ceci dépend du mécanisme de *ramassage de l'ordure*. Dans la procédure du destructeur on peut ajouter un code exécutant des actions réglementées lors de la suppression de l'objet.

Le constructeur et le destructeur de la classe peuvent être examinés en un sens comme des *méthodes* à lui. *VB.NET* entretient également trois types de procédures:
Procédure *Sub* – une succession d'opérateurs exécutant une fonctionnalité, définis de façon univoque par un nom unique. La procédure ne retourne pas de résultat par son nom. Elle peut être appelée à l'exécution à partir de chaque point de l'application. Elle est utilisée pour le traitement d'événements. Procédure *Function* – homologue de la procédure ci-dessus à cette différence que le nom apporte un résultat et qu'elle est appelée à l'exécution par un opérateur.

Procédure *Property* – pour la lecture contrôlée et l'enregistrement de valeurs de variables.

Le champ représente une variable définie dans la classe qu'on élabore. Pour la rendre accessible par les autres classes ou objets la variable est définie comme *Public*. Pour l'affectation d'une valeur il faut créer un objet de la classe. L'affectation ou la lecture de la valeur de la variable se fait par son nom et le nom de la variable.

A la différence du champ la *propriété* est réalisée par la procédure *Property.* Aspect général de la procédure :

Private variable *As* type = valeur initiale
[Modificateur d'accès] *Property* [nom de la procédure]

Get

Return variable
End Get

*Set (*argument d'entrée par valeur*)*

variable = argument d'entrée
End Set

End Property

Les modificateurs sont *Public, Private, Protected, Friend* ou *Protected Friend. Public* est par défaut. Si on définit *Private* comme modificateur d'accès, d'autres classes n'ont pas accès aux membres de la classe qu'on élabore. Grâce aux modificateurs on effectue *l'abstraction* et le *capsulage* dans les classes.

Le nom de la procédure est suit les règles de nom de variable dans VB.NET.

La procédure *Get Property* retourne la valeur de la propriété. Pour ce faire on a besoin d'une variable supplémentaire à accès limité qui doit recevoir la valeur retournée. Il est recommandé de placer des opérateurs exécutant une fonctionnalité définie comme par exemple vérification de la valeur lue, certaines opérations de calculs, de message, de diagnostic etc. entre Get et Return.

La procédure *Set Property* est destinée à l'affectation d'une valeur de la propriété. Une variable qui est acceptée par valeur, passe comme argument de la procédure. Avant de l'affecter à la propriété il est recommandé de faire une vérification de la valeur saisie.

VB.NET permet la déclaration de propriété en lecture seule par le mot clé *ReadOnly* ou en écriture seule *WriteOnly*. Ils sont placés avant le modificateur d'accès. On peut s'attendre au même résultat si la procédure *Get* ou *Set* n'est pas dans *Property*.

Les classes dans VB.NET peuvent être *héritées* c'est-à-dire d'une classe de base on peut élaborer une classe dérivée. La classe dérivée hérite les méthodes et les propriétés de la classe de base. Ces méthodes peuvent être modifiées et complétées dans la classe dérivée. De cette manière on élargit la fonctionnalité de la classe de base dans la classe dérivée.

Lors de l'élaboration de la classe dérivée on utilise le mot clé *Inherits* pour indiquer la classe de base. Lors de la réécriture d'une méthode dans la classe dérivée on utilise *Overrides*. Elle montre que la procédure correspondante dans la classe de base sera remplacée par une nouvelle. Elle n'est pas obligatoire si la fonctionnalité de la méthode correspondante ne change pas. La structure est comme suit :

Public Class nom de la classe dérivée
    Inherits nom de la classe de base
    Déclaration de propriétés et de méthodes
    Overrides type et nom de la procédure
        Code de programmation exécutant une fonctionnalité
    End de la procédure
    Autres procédures (méthodes)
End Class

2.2.2. Espace des noms

Dans VB.NET on utilise des noms uniques pour l'identification de classes, d'objets, de collections, de méthodes, de propriétés, de variables etc. Lorsque les objets sont groupés dans un bloc assemblé, ils sont soumis à une certaine hiérarchie. Pour organiser l'hiérarchie des objets et pour simplifier l'accès on introduit la notion *espace des noms*.

Dans l'environnement NET l'espace des noms commence par System. Il comprend plus de 100 classes à application variée. Une partie d'elles représentent des types de données de base accessibles à partir des applications conçues. L'espace System et 24 espaces de noms du niveau 2 sont destinés à:

Données : *System.Data, .Xml, .Xml.Serialization*
Modèle de composante: *System.CodeDom, .ComponentModel*
Configuration: *System.Configuration*
Services: *System.Diagnostics, .DirectoryServices, .Management, .ServiceProcess, . Messaging, .Timers*

Globaliation: *System.Globalization, .Resources*
Programmation réseau: *System.NET*
Programmation: *System.Collections, .IO, .Text,*
.TextRegularExpressions,.Threading
Réflexion: *System.Reflection*
Graphique: *System.Drawing, .Windows.Forms*
Services: *System.RuntimeComputerServices,.*
RuntimeRemoting,RuntimeInteropServices,.
RuntimeSerialization
Sécurité: *System.Security*
Services Web: *System. Web, .WebServices*

A part les espaces de noms entretenus par l'environnement NET dans VB.NET on peut créer également des espaces personnels à partir de :

Namespace nom de l'espace
Code de programmation des classes
End Namespace

Les classes inclues entretiendront l'espace au nom indiqué. Par analogie les espaces de noms peuvent être placés l'un dans l'autre par l'insertion de la construction *Namespace, End Namespace* l'une dans l'autre. Lors de la deuxième insertion on peut utiliser l'espace déjà créé, par exemple

Namespace nom de base. nouveau nom
Texte de programmation de classes
End Namespace

Où le nom de base de l'espace est déjà déclaré dans une construction antérieure.

Exemple

En utilisant les méthodes et les propriétés des objets string créer une classe *FSLName* adoptant un nom bulgare dans la variable *PName* et divisant le nom en trois *FirstName*, *SecondName* et *LastName* .

Public Class FSLName

Public Sub FName*(ByRef* PName *As String, ByRef* FirsName *As String)*

```
    Dim n As Integer
    PName = PName.Trim
    FirsName = ""
    n = PName.IndexOf(" ")
    If n > 0 Then FirsName = PName.Substring(0, n)
  End Sub

  Public Sub SName (ByRef PName As String, ByRef SecondName As String)
    Dim n1 As Integer
    Dim n2 As Integer
    PName = PName.Trim
    SecondName = ""
    n1 = PName.IndexOf(" ")
    If n1 > 0 Then
      n2 = PName.IndexOf(" ", n1 + 1)
      If n2 > 0 Then SecondName = PName.Substring (n1 + 1, n2 - n1)
    End If
  End Sub

  Public Sub LName (ByRef  PName As String, ByRef  LastName As String)
    Dim n1 As Integer
    Dim n2 As Integer
    PName = PName.Trim
    LastName = ""
    n1 = PName.IndexOf(" ")
    If n1 > 0 Then
      n2 = PName.IndexOf(" ", n1 + 1)
      If n2 < PName.Length Then
        If n2 > 0 Then LastName = PName.Substring (n2 + 1)
      End If
    End If
  End Sub

End Class
```

Texte du programme appelant un objet de la classe

```
    Dim PName As String
    Dim FirsName As String
    Dim SecondName As String
    Dim LastName As String
```

*Dim **BulName** As New FSLName ()*

*PName = "*Ivan Ivanov Ivanovski*"*

***BulName**.Fname (PName, FirsName)*
***BulName**.Sname (PName, SecondName)*
***BulName**.LName (PName, LastName)*

Le programme marchera correctement uniquement si le nom saisi est composé de trois mots, séparés par un espace.

2.2.3. Exemple d'héritage et de polymorphisme

L'héritage représente la possibilité d'obtenir une dérivée à partir d'une classe de base définie possédant les méthodes et les propriétés de celle-ci. A la différence des autres langages orientés objet VB.NET entretient un seul héritage. Une classe dérivée peut hériter uniquement une seule classe de base. Celle-ci peut avoir hérité une autre classe de base etc. VB.NET ne permet pas l'héritage simultané de plusieurs classes de base par une classe dérivée.

Le polymorphisme représente une possibilité de modification de certaines méthodes de la classe de base dans la classe dérivée.

L'exemple ci-dessous contient les modifications suivantes:

1. la classe de base *FSLName* et la classe dérivée *NewFSLName* se trouvent dans un module commun *Module1*. Cette condition n'est pas obligatoire.
2. Des constructeurs du type :

```
Public Sub New ()
    MyBase.new ()
End Sub
```

sont prévus dans la classe de base et la classe dérivée.

3. Dans la procédure de la classe de base qui doit être modifiée est ajouté le mot clé *Overridable.* Ceci représente une autorisation de réécriture de la méthode correspondante dans la classe dérivée.

*Public **Overridable** Sub LName (ByRef PName As String, ByRef LastName As String)*

4. Dans la classe dérivée on a ajouté, à part le constructeur *MyBase.new ()* le mot clé *Inherits* qui indique la classe de base.

Inherits FSLName

5. La procédure *LName* est modifiée de la façon suivante :

```
Overrides Sub LName (ByRef PName As String, ByRef LastName As
String)
        Dim n1 As Integer
        Dim n2 As Integer
        PName = PName.Trim
        LastName = ""
        n1 = PName.IndexOf(" ")
        If n1 > 0 Then
           n2 = PName.IndexOf(" ", n1 + 1)
           If n2 < PName.Length Then
              If n2 > 0 Then
                 LastName = PName.Substring(n2 + 1)
                 LastName = LastName.ToUpper
              End If
           End If
        End If
     End Sub
```

La nouvelle fonctionnalité représente un opérateur transformant les minuscules et majuscules.

6. Dans le menu principal de l'environnement IDE par Project| Add Reference la référence suivante est ajoutée

System.Windows.Forms.DLL

7. Dans le programme d'appel on a fait les corrections suivantes :

```
Dim BulName As FSLName
BulName = New NewFSLName ()
```

BulName.LName retourne valeur à *LastName* = "IVANOVSKI"

Lors de la création de l'objet *BulName* de la classe dérivée *NewFSLName* on appelle son constrcuteur. Initialement il exécute le constructeur de la classe de base. La procédure *LName* est remplacée par homonyme de la classe dérivée.

Si l'on exécute l'opérateur

BulName = New FSLName ()

Lors des appels ultérieurs de *BulName* les méthodes de la classe de base seront exécutées.