Tables des matières des pages VBPA

L'espace de noms System.Collections

Une collection regroupe un ensemble d'objets quelconques, comme le fait un tableau, mais avec une notion d'indice moins marquée. Il est ordinaire d'ajouter et de retirer des éléments d'une collection sans en connaître la position.
Tous les tableaux sont dérivés d'une classe **`Array`** et les collections sont dérivées de celles proposées dans **`System.Collections`**. Un exemple trivial de différence entre le tableau et la collection réside dans le moyen de connaître le nombre d'éléments : il est donné par la propriété **`Length`** d'un tableau et par la propriété **`Count`** d'une collection. La collection peut contenir des objets de types originels différents (entiers, chaînes, tableaux, autres collections, objets divers, …) alors que tous les éléments d'un tableau sont d'un même type.
De plus, contrairement aux tableaux, le programmeur doit rarement se soucier de la taille d'une collection. L'espace de noms **`System.Collections`** fournit quelques classes directement opérationnelles ainsi qu'une classe **`CollectionBase`** offrant les outils de base d'une collection et permettant par héritage la construction de toute autre collection.
Enfin, l'espace de noms **`System.Collections`** possède un sous espace **`System.Collections.Generic`** qui propose plusieurs sortes de collections pouvant être instanciées pour constituer une alternative à l'usage des tableaux. En effet, comme c'est le cas des tableaux, ces collections doivent être instanciées pour un type prédéfini de données. Elles permettent donc d'allier les avantageuses fonctionnalités des collections à la rigueur des structures fortement typées. L'espace de noms **`System.Collections.Generic`** est une nouveauté depuis VS 2005.

Les classes offertes

`ArrayList`	C'est la plus simple des collections. L'accès à un élément se fait essentiellement par son index.
`BitArray`	Il s'agit d'un tableau de valeurs booléennes stockées à raison d'un bit par valeur. Une valeur est accessible par son index.
`Hashtable`	C'est une collection de paires objet-clé. L'accès à un élément se fait essentiellement par sa clé.
`SortedList`	Il s'agit d'une version améliorée de la **`Hashtable`** dans laquelle les éléments sont triés selon la valeur de leur clé.
`Queue`	Comme son nom l'indique, il s'agit d'une collection dans laquelle on ajoute ou retire un élément selon le principe de la file (FIFO : premier entré, premier sorti).
`Stack`	Comme son nom l'indique, il s'agit d'une collection dans laquelle on ajoute ou retire un élément selon le principe de la pile (LIFO : dernier entré, premier sorti).

Membres communs à la plupart des collections

Les exemples de programmations relatifs aux membres communs illustrent dans les pages suivantes, l'étude des classes énumérées ci-dessus.

Propriétes

`Count`	Nombre d'éléments de la collection.
`IsFixedSize`	Cette propriété possède la valeur **`True`** ou **`False`** selon que la collection est de taille fixe ou non.
`IsReadOnly`	Cette propriété possède la valeur **`True`** ou **`False`** selon que la liste est en lecture seul ou non.
`Item`	L'**`Item`** fournit l'élément dont on lui a indiqué l'index ou la clé.

Méthodes

`Add`	Ajoute un élément à une collection.
`Clear`	Vide la collection de tout son contenu.
`Clone`	Crée une copie partielle d'une collection.
`Contains`	Indique si un élément donné est présent dans la collection.
`CopyTo`	Copie partielle ou totale d'une collection dans un tableau à une dimension.

La classe ArrayList

Propriété

Capacity

Cette propriété représente le nombre d'éléments que l'**ArrayList** est capable de stocker. Il ne faut pas confondre cette propriété avec **Count** qui représente le nombre réel d'éléments stockés. La valeur de **Capacity** est toujours supérieure ou égale à **Count**. Si **Count** excède **Capacity** lors de l'ajout d'éléments, la capacité de la liste est doublée par une réallocation automatique du tableau interne.

Méthodes

AddRange

Ajout d'une collection **ArrayList** à la fin de l'**ArrayList**.

```
Dim A As New ArrayList                          ' Instanciation d'un ArrayList
Dim B As New ArrayList
Dim I As Integer
Dim S As String
Dim Obj As Object
I = 5
S = "AZERTY"
A.Add(I)                                        ' Stocke un entier dans A
Console.WriteLine("A " & A.Item(0))             ' Affiche A 5
A.Add(S)                                        ' Stocke une chaîne dans A
Console.WriteLine("A " & A.Item(1))             ' Affiche A AZERTY
B.Add(I * 2)                                    ' Stocke un entier dans B
Console.WriteLine("B " & B.Item(0))             ' Affiche B 10
B.Add("B" & S)                                  ' Stocke une chaîne dans B
Console.WriteLine("B " & B.Item(1))             ' Affiche B BAZERTY
A.AddRange(B)                                   ' Stocke l' ArrayList B dans A
For Each Obj In A
    Console.WriteLine("A = " & Obj)             ' Affiche A = 5
Next                                            '         A = AZERTY
                                                '         A = 10
                                                '         A = BAZERTY
```

GetType

Obtient le type de toute instance ou d'un seul élément.

```
Console.WriteLine(B.GetType)                    ' Affiche System.Collections.ArrayList
Console.WriteLine(B.Item(1).GetType)            ' Affiche System.String
```

Insert

Insère un élément dans **ArrayList** à l'index spécifié et provoque le déplacement de tous les autres éléments d'une position vers la fin.

```
B.Insert(1, "Insertion")
Console.WriteLine(B.Item(0))                    ' Affiche 10
Console.WriteLine(B.Item(1))                    ' Affiche Insertion
Console.WriteLine(B.Item(2))                    ' Affiche BAZERTY
```

InsertRange

Insère les éléments d'une collection **ArrayList** à l'index spécifié et provoque le déplacement de tous les autres éléments vers la fin.

```
            ' Contenu de A avant InsertRange : 5 AZERTY 10 BAZERTY
A.InsertRange(1, B)
            ' Contenu de A après InsertRange : 5 10 Insertion BAZERTY AZERTY 10 BAZERTY
```

IndexOf

Retourne l'index de base zéro de la première occurrence d'une valeur dans **ArrayList** ou dans une partie de celui-ci, ou **-1** en cas d'échec de la recherche.

```
Console.WriteLine(A.IndexOf(10))              ' Affiche 1
```

LastIndexOf

Retourne l'index de base zéro de la dernière occurrence d'une valeur dans **ArrayList** ou dans une partie de celui-ci, ou **-1** en cas d'échec de la recherche.

```
Console.WriteLine(A.LastIndexOf(10))          ' Affiche 5
```

Remove

Supprime la première occurrence d'un objet spécifique de **ArrayList** et provoque le déplacement de tous les éléments suivants d'une position vers le début.

```
            ' Contenu de A avant Remove : 5 10 Insertion BAZERTY AZERTY 10 BAZERTY
A.Remove("BAZERTY")
            ' Contenu de A après Remove : 5 10 Insertion AZERTY 10 BAZERTY
```

RemoveAt

Supprime l'élément au niveau de l'index spécifié de **ArrayList** et provoque le déplacement de tous les éléments suivants d'une position vers le début.

```
            ' Contenu de A avant RemoveAt : 5 10 Insertion AZERTY 10 BAZERTY
A.RemoveAt(4)
            ' Contenu de A après RemoveAt : 5 10 Insertion AZERTY BAZERTY
```

RemoveRange

Supprime un nombre d'éléments donné de **ArrayList** à partir de l'index également donné et provoque le déplacement de tous les éléments suivants vers le début. La syntaxe est **RemoveRange(Index, Nombre)**.

```
            ' Contenu de A avant RemoveRange : 5 10 Insertion AZERTY BAZERTY
A.RemoveRange(1, 3)
            ' Contenu de A après RemoveRange : 5 BAZERTY
```

Repeat

Initialise une **ArrayList** en recopiant un élément donné un nombre de fois également donné. L'**ArrayList** est préalablement vidée de son contenu. La syntaxe est **Repeat(Objet, Quantité)**.

```
            ' Contenu de A avant Repeat : 5 10 Insertion AZERTY BAZERTY
A = ArrayList.Repeat(0, 3)
            ' Contenu de A après Repeat : 0 0 0
```

Reverse

Inverse l'ordre des éléments dans **ArrayList** ou dans une partie de celui-ci. La syntaxe est **Reverse()** ou **Reverse(Index, Nombre)**.

```
            ' Contenu de A avant Reverse : 5 10 Insertion AZERTY BAZERTY
A.Reverse()
            ' Contenu de A après Reverse : BAZERTY AZERTY Insertion 10 5
```

Sort

Trie les éléments dans **ArrayList**. Si tous les éléments de la collection sont des objets ou s'ils appartiennent à des types de bases différents, il faut indiquer la fonction de comparaison à utiliser. Ce point sera abordé plus tard.
Si les éléments sont tous d'un même type de base, la syntaxe est **Sort()**.

SetRange

Copie les éléments d'une collection sur une plage d'éléments d'un **ArrayList** à partir d'un index donné. Cette opération écrase les éléments éventuellement existants. La syntaxe est **SetRange(Index, Collection)**.

```
            ' Contenu de A avant SetRange : 5 10 15 20 17 25 30
            ' Contenu de B avant SetRange : BA BX BW
A.SetRange(2, B)
            ' Contenu de A après SetRange : 5 10 BA BX BW 25 30
            ' Contenu de B après SetRange : BA BX BW
```

CopyTo

Copie les éléments de **ArrayList** vers un tableau unidimensionnel. Ceci n'est possible que si tous les éléments de la collection et le tableau cible sont de même type. Ainsi, s'il faut copier **l'ArrayList A** de l'exemple précédent dans un tableau de chaînes, il faut d'abord procéder à la conversion des valeurs numériques en chaînes. Au programmeur de savoir quelles sont les conséquences de cette conversion de la collection dans son application.

```
Dim Resultat(10) As String
' …
For I = 0 To A.Count - 1
    A.Item(I) = A.Item(I).ToString
Next
A.CopyTo(Resultat)
```

Clone

Copie les éléments d'un **ArrayList** vers un autre. Cette opération écrase les éléments éventuellement existants dans l'**ArrayList** cible. La méthode **Clone** ne pas copie pas les objets référencés, mais seulement leurs références. Après la copie, une référence d'une collection et son homologue de l'autre pointent toutes les deux la même instance.

```
A.Add(0)                                    ' Stocke l'entier 0 dans A
Dim T = New Integer() {1, 2}                ' Crée un tableau T(0) = 1 et T(1) = 2
A.Add(T)                                    ' Stocke le tableau T dans A
B = A.Clone()                               ' Copie de A sur B
Console.WriteLine("A = " & A.Item(0))       ' Affiche A = 0
Console.WriteLine("A = " & A.Item(1)(0))    ' Affiche A = 1
Console.WriteLine("A = " & A.Item(1)(1))    ' Affiche A = 2
Console.WriteLine("B = " & B.Item(0))       ' Affiche B = 0
Console.WriteLine("B = " & B.Item(1)(0))    ' Affiche B = 1
Console.WriteLine("B = " & B.Item(1)(1))    ' Affiche B = 2
T(1) = 99                                   ' Change T(1) de 2 en 99
Console.WriteLine("A = " & A.Item(0))       ' Affiche A = 0
Console.WriteLine("A = " & A.Item(1)(0))    ' Affiche A = 1
Console.WriteLine("A = " & A.Item(1)(1))    ' Affiche A = 99
Console.WriteLine("B = " & B.Item(0))       ' Affiche B = 0
Console.WriteLine("B = " & B.Item(1)(0))    ' Affiche B = 1
Console.WriteLine("B = " & B.Item(1)(1))    ' Affiche B = 99
```

ToArray

Copie les éléments de **ArrayList** vers un nouveau tableau de type **Object**.

```
Dim T(3) As Object
A.Add(0)
A.Add(1)
A.Add("AZE")
T = A.ToArray()
For Each Obj In T
    Console.WriteLine("T = " & Obj)          ' Affiche 0 1 AZE
Next
```

Contains

Cette méthode retourne **True** si l'objet désigné est présent dans la collection et **False** dans le cas contraire.

```
Console.WriteLine(A.Contains("AZE").ToString) ' Affiche True
```

La classe BitArray

Une instance de **BitArray** est une collection d'un nombre prédéterminé de valeurs booléennes. Cette collection a peu de points communs avec les autres mais possède néanmoins les membres déjà étudiés **Count**, **IsReadOnly**, **GetType**, **Clone**, **CopyTo**, **Equals** et **ToString**.

Propriété

Length

Cette propriété représente le nombre d'éléments que **BitArray** est capable de stocker. Elle est identique à la propriété **Count**, sauf que cette dernière est en lecture seule, tandis que **Length** peut être affectée en cours d'exécution pour modifier la taille de la collection.

Méthodes

Set et SetAll

La collection **BitArray** ne possède pas de méthode **Add** du fait que tous ses éléments existent dès son instanciation. Il est toutefois nécessaire d'affecter les valeurs des éléments selon les impératifs des applications. C'est ce que permettent les méthodes **Set** et **SetAll**. Il est également possible d'affecter un élément désigné par son index et la propriété **Item**.

```
Dim A As New BitArray(8)                     ' Instanciation d'un BitArray
A.SetAll(False)                              ' Initialise tout à False
A.Set(0, True)                               ' Initialise à True à l'index 0
A.Item(6) = True                             ' Initialise à True à l'index 6
```

Get

Méthode de lecture d'un élement, **Get** retourne **True** ou **False** selon la valeur de l'élément à un index donné. La lecture de la propriété **Item** produit le même résultat.

```
Console.WriteLine(A.Get(0).ToString)         ' Affiche True
Console.WriteLine(A.Item(1).ToString)        ' Affiche False
```

Not

La méthode **Not** inverse toutes les valeurs logiques du **BitArray**. Le résultat peut être copié dans un autre **BitArray**.

```
Dim A As New BitArray(8)
Dim B As New BitArray(8)
```

```
A.SetAll(False)
B = A.Not()                                    ' Les éléments de A et B sont True
```

`And`, `Or` et `Xor`

Ces méthodes exécutent les opérations logiques correspondantes entre deux **`BitArray`** et retournent le résultat dans un autre **`BitArray`**.

```
C = A.And(B)
C = A.Or(B)
C = A.Xor(B)
```

La classe Hashtable

```
Dim A As New Hashtable                         ' Instanciation d'un Hashtable
A.Add("AZ", "AZERTi")                          ' Ajout d'éléments dans A
A.Add("QW", "QWERTY")
A.Add("PA", "PANZANI")
For Each Obj As Object In A.Keys
    Console.WriteLine(Obj.ToString)            ' Affichage AZ QW PA
Next
For Each Obj As Object In A.Values
    Console.WriteLine(Obj.ToString)            ' Affichage AZERTi QWERTY PANZANI
Next
A.Item("AZ") = "AZERTY"                        ' Modification de l'élément de clé AZ
Console.WriteLine(A.Item("AZ").ToString)       ' Affiche AZERTY
```

Propriétés

`Item`

Cette propriété permet la lecture et la modification d'une valeur correspondant à une clé donnée. Si les clés sont numériques et ordonnées de **`0`** à **`Count - 1`**, la propriété peut être utilisée comme dans le cas de l'**`ArrayList`**. Mais alors, à quoi bon utiliser la classe **`Hashtable`** ? L'usage de cette dernière n'a de sens que lorsqu'on utilise vraiment des clés.

```
A.Item("AZ") = "AZERTY"                        ' Modification de l'élément de clé AZ
```

`Keys`

Cette propriété contient la collection des clés associées aux valeurs de **`Hashtable`**.

```
For Each Obj As Object In A.Keys
    Console.WriteLine(Obj.ToString)            ' Affichage PA QW AZ
Next
```

`Values`

Cette propriété contient la collection des valeurs associées aux clés de **`Hashtable`**.

```
For Each Obj As Object In A.Values
    Console.WriteLine(Obj.ToString)            ' Affichage PANZANI QWERTY AZERTi
Next
```

Méthodes

`Add`

Ajoute un élément avec la clé et la valeur spécifiées dans **`Hashtable`**.

```
A.Add("AZ", "AZERTi")                          ' Ajout d'un élément dans A
```

Contains, **ContainsKey** et **ContainsValue**

Ces trois méthodes retournent la valeur **True** si l'élément cherché est présent dans la collection, et la valeur **False** dans le cas contraire.
Les deux premières, **Contains** et **ContainsKey**, réalisent la recherche d'un élément parmi les clés de la collection, tandis que **ContainsValue** réalise la recherche parmi les valeurs.

```
Console.WriteLine(A.Contains("QW"))           ' Affiche True
Console.WriteLine(A.Contains("PANZANI"))      ' Affiche False
Console.WriteLine(A.ContainsKey("QW"))        ' Affiche True
Console.WriteLine(A.ContainsValue("PANZANI"))' Affiche True
```

Remove

Supprime l'élément ayant la clé spécifiée.

```
A.Remove("PA")
Console.WriteLine(A.ContainsValue("PANZANI"))' Affiche False
```

La classe SortedList

La classe **SortedList** présente les mêmes membres que la classe **Hashtable** et quelques méthodes qui lui sont spécifiques. La particularité de **SortedList** est de conserver tous les éléments dans l'ordre croissant des clés. Ceci implique que toutes les clés soient du même type.

```
Dim A As New SortedList                       ' Instanciation d'un SortedList
A.Add("AZ", "AZERTi")                         ' Ajout d'éléments dans A
A.Add("QW", "QWERTY")
A.Add("PA", "PANZANI")
For Each Obj As Object In A.Keys
    Console.WriteLine(Obj.ToString)           ' Affichage AZ PA QW
Next
For Each Obj As Object In A.Values
    Console.WriteLine(Obj.ToString)           ' Affichage AZERTi PANZANI QWERTY
Next
A.Item("AZ") = "AZERTY"                       ' Modification de l'élément de clé AZ
Console.WriteLine(A.Item("AZ").ToString)      ' Affiche AZERTY
```

Méthodes

IndexOfKey

Retourne l'index de base zéro de la clé spécifiée ou **-1** en cas d'échec de la recherche.

```
Console.WriteLine(A.IndexOfKey("QW"))         ' Affiche 2
```

IndexOfValue

Retourne l'index de base zéro de la première occurrence de la valeur spécifiée ou **-1** en cas d'échec de la recherche.

```
Console.WriteLine(A.IndexOfValue("PANZANI")) ' Affiche 1
```

GetByIndex

Obtient la valeur à l'index spécifié.

```
Console.WriteLine(A.GetByIndex(2))            ' Affiche QWERTY
```

GetKey

Obtient la clé à l'index spécifié.

```
Console.WriteLine(A.GetKey(1))                ' Affiche PA
```

SetByIndex

Remplace la valeur au niveau de l'index spécifié.

```
Console.WriteLine(A.IndexOfValue("PANZANI")) ' Affiche 1
A.SetByIndex(1, "RANZANI")                   ' Change PANZANI en RANZANI
Console.WriteLine(A.GetKey(1))               ' Affiche PA
Console.WriteLine(A.Item("PA").ToString)     ' Affiche RANZANI
```

RemoveAt

Supprime l'élément au niveau de l'index spécifié.

```
A.RemoveAt(1)                                ' Supprime la paire PA - RANZANI
```

GetKeyList

Retourne une collection des clés du **SortedList**.

```
For Each Obj As Object In A.GetKeyList
    Console.WriteLine(Obj.ToString)          ' Affichage AZ QW
Next
```

GetValueList

Retourne une collection des valeurs du **SortedList**.

```
For Each Obj As Object In A.GetValueList
    Console.WriteLine(Obj.ToString)          ' Affichage AZERTY QWERTY
Next
```

La classe Queue

Un objet **Queue** est une collection gérée selon le principe de la file (FIFO : premier entré, premier sorti).

Méthodes

Outre les membres **Contains**, **Count** et **Clear** communs à la plupart des collections, la classe **Queue** possède trois méthodes qui lui sont spécifiques : **Enqueue**, **Dequeue** et **Peek**.

```
Dim A As New Queue                           ' Instanciation d'un Queue
A.Enqueue("AZERTY")                          ' Enfilement d'éléments dans A
A.Enqueue("QWERTY")
A.Enqueue("PANZANI")
For Each Obj As Object In A
    Console.WriteLine(Obj.ToString)          ' Affichage AZERTY QWERTY PANZANI
Next
```

Enqueue

La classe **Queue** ne possède pas de méthode **Add**. C'est la méthode **Enqueue** qui la remplace.

Peek

La méthode **Peek** retourne la valeur du premier élément disponible, c'est-à-dire le plus ancien de la collection sans le supprimer de la collection.

```
Console.WriteLine(A.Count.ToString)          ' Affiche 3
Console.WriteLine(A.Peek.ToString)           ' Affiche AZERTY
Console.WriteLine(A.Count.ToString)          ' Affiche encore 3
Console.WriteLine(A.Peek.ToString)           ' Affiche encore AZERTY
Console.WriteLine(A.Count.ToString)          ' Affiche toujours 3
```

Dequeue

La méthode **Dequeue** retourne la valeur du premier élément disponible, c'est-à-dire le plus ancien de la collection et le supprime de la collection.

```
Console.WriteLine(A.Count.ToString)          ' Affiche 3
Console.WriteLine(A.Dequeue.ToString)        ' Affiche AZERTY
Console.WriteLine(A.Count.ToString)          ' Affiche 2
Console.WriteLine(A.Dequeue.ToString)        ' Affiche QWERTY
Console.WriteLine(A.Count.ToString)          ' Affiche 1
```

La classe Stack

Un objet **Stack** est une collection gérée selon le principe de la pile (LIFO : dernier entré, premier sorti).

Méthodes

Outre les membres **Contains**, **Count** et **Clear** communs à la plupart des collections, la classe **Stack** possède trois méthodes qui lui sont spécifiques : **Push**, **Pop** et **Peek**.

```
Dim A As New Stack                           ' Instanciation d'un Stack
A.Push("AZERTY")                             ' Empilement d'éléments dans A
A.Push("QWERTY")
A.Push("PANZANI")
For Each Obj As Object In A
    Console.WriteLine(Obj.ToString)          ' Affichage PANZANI QWERTY AZERTY
Next
```

Push

La classe **Stack** ne possède pas de méthode **Add**. C'est la méthode **Push** qui la remplace.

Peek

La méthode **Peek** retourne la valeur du premier élément disponible, c'est-à-dire le plus récent de la collection sans le supprimer de la collection.

```
Console.WriteLine(A.Count.ToString)          ' Affiche 3
Console.WriteLine(A.Peek.ToString)           ' Affiche PANZANI
Console.WriteLine(A.Count.ToString)          ' Affiche encore 3
Console.WriteLine(A.Peek.ToString)           ' Affiche encore PANZANI
Console.WriteLine(A.Count.ToString)          ' Affiche toujours 3
```

Pop

La méthode **Pop** retourne la valeur du premier élément disponible, c'est-à-dire le plus récent de la collection et le supprime de la collection.

```
Console.WriteLine(A.Count.ToString)          ' Affiche 3
Console.WriteLine(A.Pop.ToString)            ' Affiche PANZANI
Console.WriteLine(A.Count.ToString)          ' Affiche 2
Console.WriteLine(A.Pop.ToString)            ' Affiche QWERTY
Console.WriteLine(A.Count.ToString)          ' Affiche 1
```

Une dernière comparaison

```
' ArrayList
        Dim A As New ArrayList
        A.Add("AZERTY")
        A.Add("QWERTY")
        A.Add("RANZANI")
        A.Add("PANZANI")
        For Each Obj As Object In A             ' Affichage en ordre d'encodage :
            Console.WriteLine(Obj.ToString)     '            AZERTY QWERTY RANZANI PANZANI
        Next
       '
' BitArray
        Dim B As New BitArray(4)
        B.Set(0, True)
        B.Set(1, True)
        B.Set(2, False)
        B.Set(3, True)
        For Each Obj As Object In B             ' Affichage en ordre d'encodage :
            Console.WriteLine(Obj.ToString)     '            True True False True
        Next

' Hashtable                                     ' Affichage des clés et valeurs en ordre
        Dim C As New Hashtable                  ' d'encodage depuis VS 2005, mais en ordre
        C.Add("AZ", "AZERTY")                   ' inverse sous VS 2003 …
        C.Add("QW", "QWERTY")
        C.Add("RA", "RANZANI")
        C.Add("PA", "PANZANI")
        For Each Obj As Object In C.Keys        ' Affichage en ordre d'encodage :
            Console.WriteLine(Obj.ToString)     '            AZ QW RA PA
        Next
        For Each Obj As Object In C.Values      ' Affichage en ordre d'encodage :
            Console.WriteLine(Obj.ToString)     '            AZERTY QWERTY RANZANI PANZANI
        Next

' SortedList
        Dim D As New SortedList
        D.Add("AZ", "AZERTY")
        D.Add("QW", "QWERTY")
        D.Add("RA", "RANZANI")
        D.Add("PA", "PANZANI")
        For Each Obj As Object In D.Keys        ' Affichage en ordre croissant des clés :
            Console.WriteLine(Obj.ToString)     '            AZ PA QW RA
        Next
        For Each Obj As Object In D.Values      ' Affichage en ordre croissant des clés :
            Console.WriteLine(Obj.ToString)     '            AZERTY PANZANI QWERTY RANZANI
        Next

' Queue
        Dim E As New Queue
        E.Enqueue("AZERTY")
        E.Enqueue("QWERTY")
        E.Enqueue("RANZANI")
        E.Enqueue("PANZANI")
        For Each Obj As Object In E             ' Affichage en ordre d'encodage :
            Console.WriteLine(Obj.ToString)     '            AZERTY QWERTY RANZANI PANZANI
        Next

' Stack
        Dim F As New Stack
        F.Push("AZERTY")
        F.Push("QWERTY")
        F.Push("RANZANI")
        F.Push("PANZANI")
        For Each Obj As Object In F             ' Affichage en ordre inverse de l'encodage :
            Console.WriteLine(Obj.ToString)     '            PANZANI RANZANI QWERTY AZERTY
        Next
```

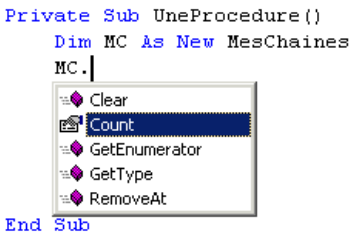
La classe CollectionBase

Depuis le début de ce cours, les propriétés et méthodes les plus utilisées des classes les plus exploitées ont été étudiées. Tous ces membres étudiés ont une particularité commune : ils sont tous membres **Public** des classes qui les exposent. Les programmeurs qui ont conçu ces classes ont bien du utiliser aussi des variables locales et des membres **Private** ou **Friend**, mais nul n'en saura jamais rien, à moins que leurs sources soient publiées … .
Chaque classe expose également des membres qui n'ont pas été repris dans les énumérations du cours, ce sont les membres **Protected**. Le programmeur souhaitant dériver des classes du Framework a tout intérêt à consulter l'aide de VB.Net pour connaître les membres **Protected** dont il peut disposer.
Tous les membres **Public** des classes sont accessibles à partir de toute instance. Les membres **Protected** quant à eux, ne sont accessibles que par héritage dans des classes dérivées. Une classe dérivée dispose alors des membres **Public** et des membres **Protected** de la classe dont elle hérite.
La classe **CollectionBase** expose peu de membres **Public**, pas assez pour gérer la moindre collection. Elle est spécialement destinée à la création de collections inédites et c'est au programmeur d'implémenter les outils et membres nécessaires.

Pour illustrer l'emploi de **CollectionBase**, une collection **MesChaines** est créée. Elle présente la particularité de la **SortedList** de conserver ses éléments en ordre alliée à la simplicité de l'**ArrayList**. La collection **MesChaines** ne peut contenir que des objets de type **String** qui ne sont pas associés à des clés et qui sont toujours dans le bon ordre. Cette programmation qui est illustrative, n'a aucune prétention de performance.

```
Class MesChaines
    Inherits CollectionBase
End Class
```

Avec ce code, la nouvelle classe **MesChaines** existe mais son exploitation dans du code client reste fort limitée. Elle ne dispose en effet que de quatre méthodes et d'une propriété. Elle ne dispose d'aucun moyen d'ajouter des éléments à la collection, ni d'en extraire.



Il faut donc implémenter une méthode **Add** et une propriété **Item**, ainsi que les moyens souhaités pour la classe **MesChaines**. Voici le code complet de cette nouvelle classe.

```
Class MesChaines
  Inherits CollectionBase                              ' Héritage de CollectionBase
  Dim Sens As Integer                                  ' Sens du tri : croissant/décroissant
  Dim FlgTri As Boolean

  Public Sub New(Optional ByVal Ordre As Integer = 1)  ' Un constructeur est programmé pour
    Sens = Ordre                                       ' prendre en compte un paramètre
  End Sub                                              ' indiquant le sens du tri.

  Public ReadOnly Property SensTri() As String
    Get
      If Sens = -1 Then                                ' La propriété SensTri en ReadOnly
        Return "D"                                     ' permet au code client de connaître
      Else                                             ' le paramètrage actuel du tri.
        Return "C"                                     ' La propriété SensTri vaut D si le
      End If                                           ' tri est décroissant ou C s'il est
    End Get                                            ' croissant.
  End Property

  Public Property Item(ByVal Index As Integer) As String
    Get
      Return MyBase.InnerList(Index)                   ' La propriété Item permet la lecture
    End Get                                            ' et l'écriture de l'élément dont
    Set(ByVal Value As String)                         ' l'index lui est passé.
      MyBase.Innerlist(Index) = Value                  ' Cette propriété gère la propriété
      If Not FlgTri Then Tri()                         ' Innerlist reçue par héritage.
    End Set                                            ' FlgTri empêche l'appel récursif du
  End Property                                         ' tri quand Item y est modifié.
```

```
  Public Sub Add(ByVal Obj As String)
    MyBase.InnerList.Add(Obj)                          ' La méthode Add permet l'ajout
    Tri()                                              ' d'un élément à la collection et
  End Sub                                              ' enclenche le tri.

  Public Sub ChangeOrdre(Optional ByVal Ordre As Integer = 0)
    If Ordre < -1 Or Ordre > 1 Then Ordre = 0
    If Ordre <> Sens Then                              ' La méthode ChangeOrdre permet au
      Sens = Sens * -1                                 ' code client d'inverser l'ordre du
      Tri()                                            ' tri.
    End If
  End Sub

  Private Sub Tri()
    Dim i As Integer                                   ' La procédure privée Tri réalise le
    Dim Comparaison As Integer                         ' tri de la collection.
    Dim Temp As String                                 ' L'algorithme de tri utilisé ici
    Dim TemoinPermut As Byte                           ' n'est autre qu'une variante du tri
    If Count < 2 Then Exit Sub                         ' à bulles.
    FlgTri = True
    Do
      TemoinPermut = 0
      For i = 0 To Count - 2
        Comparaison = String.Compare(Item(i + 1), Item(i))
        If Comparaison <> Sens And Comparaison <> 0 Then
          Temp = Item(i)
          Item(i) = Item(i + 1)                        ' Il y a ici affectation de la
          Item(i + 1) = Temp                           ' propriété Item de la collection.
          TemoinPermut = 1
        End If
      Next
    Loop While TemoinPermut = 1
    FlgTri = False
  End Sub
End Class
```

Utilisation de la classe **MesChaines** dans une application cliente :

```
        Dim MC As New MesChaines                       ' Instanciation d'un MesChaines
        Console.WriteLine(MC.SensTri().ToString)       ' Affiche C
        MC.Add("AZERT")                                ' Ajout d'un élément dans MC
        MC.Add("QWERT")
        MC.Add("RANZA")
        MC.Add("PANZA")
        For Each S As String In MC
            Console.WriteLine(S)                       ' Affichage AZERT PANZA QWERT RANZA
        Next
        MC.ChangeOrdre()                               ' Inverse l'ordre de tri
        For Each S As String In MC
            Console.WriteLine(S)                       ' Affichage RANZA QWERT PANZA AZERT
        Next
```

Autre exemple :

```
        Dim MC As New MesChaines(-1)                   ' Instanciation pour tri décroissant
        Console.WriteLine(MC.SensTri().ToString)       ' Affiche D
        MC.Add("AZERT")                                ' Ajout d'un élément dans MC
        MC.Add("QWERT")
        MC.Add("RANZA")
        MC.Add("PANZA")
        MC.Item(1) = "Quartz"                          ' Change QWERT en Quartz
        For Each S As String In MC
            Console.WriteLine(S)                       ' Affichage RANZA Quartz PANZA AZERT
        Next
```

L'espace de noms Collections.Generic

L'espace de noms **`System.Collections.Generic`** fournit des classes de base pour la mise en exploitation de collections dont les données sont d'un type prédéfini choisi parmi les types standard du Framework. L'utilité évidente de ces classes est d'offrir une alternative à l'usage des tableaux.
L'utilisation d'une collection à la place d'un tableau présente notamment comme avantage d'être dynamique de façon transparente pour le programmeur, ou encore de disposer d'une propriété **`Count`** qui contient le nombre d'éléments effectivement présents dans la collection alors que la propriété **`Length`** d'un tableau contient le nombre de positions réservées, mais pas le nombres de positions effectivement utilisées.

Voici quelques unes des collections **`Generic`** proposées :

`List`	Liste de valeurs accessibles par leur index. Cette collection dispose des méthodes de recherche, de tri et de manipulation de listes, comme l'**`ArrayList`**.
`Dictionary`	Collection de clés et de valeurs, comme la **`HashTable`**.
`SortedDictionary`	Collection des paires clé-valeur qui sont triées sur la clé, comme la **`SortedList`**.
`LinkedList`	Liste doublement chaînée offrant notamment des méthodes de parcours **`Previous`** et **`Next`** ainsi que des possibilités d'insertion de types *First*, *Last*, *Before* et *After*.
`Queue`	Collection d'objets FIFO, comme la **`Collections.Queue`**, avec les mêmes méthodes.
`Stack`	Collection de type LIFO, comme la **`Collections.Stack`**, avec les mêmes méthodes.

Quelques exemples.

```
' List (Of String)
      Dim Ls As New Collections.Generic.List(Of String)
      Ls.Add("AZERTY")
      Ls.Add("QWERTY")
      Ls.Add("RANZANI")
      Ls.Add("PANZANI")
      For Each Obj As Object In Ls              ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            AZERTY QWERTY RANZANI PANZANI
      Next

' List (Of Integer)
      Dim Li As New Collections.Generic.List(Of Integer)
      Li.Add(456)
      Li.Add(123)
      Li.Add(789)
      Li.Add(258)
      For Each Obj As Object In Li              ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            456 123 789 258
      Next

' Dictionary (Keys String, Values Integer)
      Dim Dsi As New Collections.Generic.Dictionary(Of String, Integer)
      Dsi.Add("AZ", 456)
      Dsi.Add("QW", 123)
      Dsi.Add("RA", 789)
      Dsi.Add("PA", 258)
      For Each Obj As Object In Dsi.Keys        ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            AZ QW RA PA
      Next
      For Each Obj As Object In Dsi.Values      ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            456 123 789 258
      Next
```

```
' Dictionary (Keys Integer, Values String)
      Dim Dis As New Collections.Generic.Dictionary(Of Integer, String)
      Dis.Add(456, "AZERTY")
      Dis.Add(123, "QWERTY")
      Dis.Add(789, "RANZANI")
      Dis.Add(258, "PANZANI")
      For Each Obj As Object In Dis.Keys        ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            456 123 789 258
      Next
      For Each Obj As Object In Dis.Values      ' Affichage en ordre d'encodage :
        Console.WriteLine(Obj.ToString)         '            AZERTY QWERTY RANZANI PANZANI
      Next

' SortedDictionary (Keys Integer, Values String)
      Dim SDis As New Collections.Generic.SortedDictionary(Of Integer, String)
      SDis.Add(456, "AZERTY")
      SDis.Add(123, "QWERTY")
      SDis.Add(789, "RANZANI")
      SDis.Add(258, "PANZANI")
      For Each Obj As Object In SDis.Keys       ' Affichage en ordre croissant des clés :
        Console.WriteLine(Obj.ToString)         '            123 258 456 789
      Next
      For Each Obj As Object In SDis.Values     ' Affichage en ordre croissant des clés :
        Console.WriteLine(Obj.ToString)         '            QWERTY PANZANI AZERTY RANZANI
      Next

' LinkedList
      Dim LKs As New Collections.Generic.LinkedList(Of String)
      Dim NoeudAPlacer As LinkedListNode(Of String)
      Dim NoeudCible As LinkedListNode(Of String)

      NoeudAPlacer = New LinkedListNode(Of String)("AZER")
      LKs.AddFirst(NoeudAPlacer)

      NoeudAPlacer = New LinkedListNode(Of String)("RANA")
      LKs.AddLast(NoeudAPlacer)

      NoeudCible = LKs.FindLast("AZER")
      NoeudAPlacer = New LinkedListNode(Of String)("KIWI")
      LKs.AddAfter(NoeudCible, NoeudAPlacer)    ' Attention : erreur si NoeudCible = Nothing

      NoeudCible = LKs.FindLast("KIWI")
      NoeudAPlacer = New LinkedListNode(Of String)("KAWA")
      LKs.AddBefore(NoeudCible, NoeudAPlacer)

      NoeudAPlacer = New LinkedListNode(Of String)("PANA")
      LKs.AddAfter(NoeudCible, NoeudAPlacer)

      NoeudAPlacer = New LinkedListNode(Of String)("ERGO")
      LKs.AddFirst(NoeudAPlacer)

      For Each Obj As Object In LKs             ' Affichage en ordre attendu :
        Console.WriteLine(Obj.ToString)         '            ERGO AZER KAWA KIWI PANA RANA
      Next
```

Les types de données des collections génériques peuvent être des types construits par le programmeur. Voici pour exemple une **LinkedList** de **Personne**, le type **Personne** étant une structure.

```
      Private Structure Personne
        Dim NomPers As String
        Dim AgePers As Byte
      End Structure

      Dim MaListe As New Collections.Generic.LinkedList(Of Personne)
      Dim NoeudAPlacer As LinkedListNode(Of Personne)
      Dim NoeudCible As LinkedListNode(Of Personne)
```

```
Dim UnePersonne As Personne

UnePersonne.NomPers = "AZER"
UnePersonne.AgePers = 22
NoeudAPlacer = New LinkedListNode(Of Personne)(UnePersonne)
MaListe.AddFirst(NoeudAPlacer)

UnePersonne.NomPers = "RANA"
UnePersonne.AgePers = 66
NoeudAPlacer = New LinkedListNode(Of Personne)(UnePersonne)
MaListe.AddLast(NoeudAPlacer)

UnePersonne.NomPers = "AZER"
UnePersonne.AgePers = 22
NoeudCible = MaListe.FindLast(UnePersonne)      ' Recherche d'un noeud

If NoeudCible IsNot Nothing Then                ' Si le noeud est trouvé
  UnePersonne.NomPers = "KIWI"
  UnePersonne.AgePers = 44
  NoeudAPlacer = New LinkedListNode(Of Personne)(UnePersonne)
  MaListe.AddAfter(NoeudCible, NoeudAPlacer)
End If
```