

Les classes conteneurs.

I) Présentation.

Les classes conteneurs sont des classes qui mettent en œuvre des tableaux dynamiques (ajustement automatique de la taille), des piles, des listes chaînées, etc.

Un objet conteneur contient des objets organisés d'une manière ou d'une autre.

Il existe deux types de conteneurs:

- Les conteneurs qui héberge des objets dérivés de la classe de base Object, donc n'importe quel type d'objets.
- Les conteneurs génériques (templates).

II) Les conteneurs d'objets.

Ces conteneurs peuvent contenir n'importe quel type d'objet. Un casting (transtypage) est nécessaire lors du retrait d'un élément du conteneur, mais pas à l'introduction.

1) Les tableaux dynamiques.

A) La classe ArrayList

Classe ArrayList		
ArrayList ← Object		
using System.Collections;		
Propriétés de la classe ArrayList		
Capacity	int	Nombre d'éléments que peut contenir le tableau. Comme le tableau s'agrandit automatiquement, cette propriété présente peu d'intérêt. Vous pourriez néanmoins spécifier une taille de tableau en initialisant cette propriété.
Count	int	Nombre d'éléments présents dans le tableau.
Méthodes de la classe ArrayList		
ArrayList()		Constructeur.
int Add(object obj);		Ajoute l'objet obj en fin de tableau et renvoie le nombre d'éléments dans le tableau suite à l'insertion. Au besoin, la taille du tableau est augmentée, elle est en fait doublée pour éviter de trop fréquentes et coûteuses réorganisations.
void AddRange(ICollection c);		Ajoute une collection d'objets (par exemple, un tableau d'objets) dans le tableau dynamique.

<code>int BinarySearch(int index, int count, object value, IComparer comparer);</code>	Effectue une recherche dichotomique dans un tableau dynamique d'objets. La recherche est plus précisément effectuée sur les <code>count</code> objets à partir du <code>index-ième</code> . L'objet à rechercher est passé en troisième argument. Si la fonction de tri est connue (par exemple parce qu'il s'agit d'un type primitif ou parce que la classe implémente <code>IComparable</code>), <code>null</code> peut être passé en dernier argument. <code>BinarySearch</code> renvoie l'indice de l'élément trouvé ou une valeur négative si l'élément n'a pas été trouvé.	
<code>void Clear();</code>	Vide le tableau de ses éléments.	
<code>bool Contains(object obj);</code>	Renvoie true si le tableau contient l'objet <code>obj</code> .	
<code>void CopyTo(Array);</code>	Copie le contenu d'un tableau dynamique dans un tableau ordinaire non sujet à des agrandissements automatiques (<code>al</code> étant de type <code>ArrayList</code> d'objets <code>Pers</code>) : <code>Pers[] tabPers = new Pers[al.Count]; al.CopyTo(tabPers);</code>	
<code>void CopyTo(Array, int pos);</code>	Comme la fonction précédente mais copie les objets du tableau dynamique à partir de la <code>pos-ième</code> entrée dans le tableau de longueur fixe.	
<code>IEnumerator GetEnumerator();</code>	Renvoie un énumérateur pour parcourir le tableau dynamique (une collection de manière générale). La propriété et les deux méthodes de <code>IEnumerator</code> sont :	
	<code>Current</code>	donne l'objet se trouvant à la position de l'itérateur. <code>Current</code> est une propriété en lecture seule ;
	<code>bool MoveNext();</code>	déplace l'itérateur sur l'objet suivant. Un premier <code>MoveNext</code> est toujours nécessaire. <code>MoveNext</code> renvoie <code>false</code> quand la fin de la collection a été atteinte ;
	<code>void Reset();</code>	réinitialise l'itérateur à sa position initiale.
	Bien que l'utilisation des crochets (comme pour tout tableau) soit plus simple, un exemple de balayage par itérateur est donné plus loin dans ce chapitre. C# version 2 a introduit une nouvelle syntaxe pour les itérateurs (voir section 4.3).	
<code>ArrayList GetRange(int index, int count);</code>	Renvoie un <code>ArrayList</code> qui contient <code>count</code> objets (à partir du <code>index-ième</code>) du tableau dynamique sur lequel porte la fonction.	
<code>int IndexOf(Object);</code>	Renvoie l'index de l'objet passé en argument dans le tableau dynamique. <code>IndexOf</code> renvoie -1 si l'objet n'a pas été trouvé.	
<code>int IndexOf(Object, int index);</code>	Comme la fonction précédente mais la recherche démarre au <code>index-ième</code> objet du tableau dynamique.	
<code>int IndexOf(Object, int startIndex, int endIndex);</code>	Comme la fonction précédente mais la recherche n'est effectuée qu'entre <code>startIndex</code> (inclus) et <code>endIndex</code> (non inclus).	
<code>void Insert(int pos, object obj);</code>	Insère l'objet <code>obj</code> en <code>pos-ième</code> position dans le tableau.	
<code>void InsertRange(int index, ICollection);</code>	Insère une collection d'objets dans le tableau dynamique.	
<code>int LastIndexOf(Object);</code>	Comme <code>IndexOf</code> mais la recherche est effectuée de la fin vers le début. Les autres formes de <code>IndexOf</code> s'appliquent à <code>LastIndexOf</code> .	

Méthodes de la classe ArrayList (suite)

<code>void Remove(object obj);</code>	Retire l'objet obj du tableau.
<code>void RemoveAt(int pos);</code>	Retire l'objet en <i>pos-ième</i> position. Les éléments qui suivent « avancent » d'une position. L'exception <code>ArgumentOutOfRangeException</code> est générée si ce <i>pos-ième</i> élément n'existe pas.
<code>void RemoveRange(int pos, int nb);</code>	Retire nb objets à partir de la <i>pos-ième</i> position.
<code>ArrayList Repeat(object obj, int nb);</code>	Méthode statique qui renvoie un objet <code>ArrayList</code> contenant nb fois l'objet obj.
<code>void Reverse();</code>	Inverse l'ordre des éléments dans le tableau.
<code>void Reverse(int index, int count);</code>	Inverse l'ordre des count objets à partir du <i>index-ième</i> .
<code>void Sort();</code>	Trie le tableau. Voir exemple plus loin dans ce chapitre.
<code>void Sort(IComparer);</code>	Trie le tableau en spécifiant la classe effectuant la comparaison. Voir exemple plus loin dans ce chapitre.
<code>void Sort(int index, int count, IComparer);</code>	Même chose mais trie count objets à partir du <i>index-ième</i> . Le dernier argument peut être null si la méthode de comparaison n'est pas nécessaire (cas d'un tri d'entiers, de réels ou de chaînes de caractères car l'ordre de ces éléments est connu).
<code>object[] ToArray();</code>	Renvoie un tableau (de taille fixe) d'objets à partir du tableau dynamique (voir les exemples pour comprendre le contexte) :
	<pre>object[] tabPers = a1.ToArray(); foreach (Pers p in tabPers) Console.WriteLine(p);</pre>
<code>void TrimToSize();</code>	Ajuste la taille du tableau au nombre d'éléments présents (lorsque le tableau doit être agrandi, sa taille est doublée, ce qui est parfois inutile).

- Exemple

```

using System;
using System.Collections;
public class SamplesArrayList {
    public static void Main() {
        // Creates and initializes a new ArrayList.
        ArrayList myAL = new ArrayList();
        myAL.Add("Hello");
        myAL.Add("World");
        myAL.Add("!");

        // Displays the properties and values of the ArrayList.
        Console.WriteLine("myAL");
        Console.WriteLine("Count: {0}", myAL.Count);
        Console.WriteLine("Capacity: {0}", myAL.Capacity);
        Console.Write("Values:");
        PrintValues(myAL);
    }

    public static void PrintValues( IEnumerable myList ) {
        foreach ( Object obj in myList ) {
            Console.Write("{0}", obj);
            Console.WriteLine();
        }
    }
}

/*
This code produces output similar to the following:

myAL
Count: 3
Capacity: f
Values: Hello World !
*/

```

- Exercice

Objectif:

Créer un programme pour gérer un parc informatique:

- Marque et modèle de l'ordinateur (tour ou Desktop), Taille Ecran, OS, processeur, quantité RAM, capacité disque dur, age, en panne ou pas, intégré au domaine lyc-loritz ou pas.

Moyen:

- Utiliser un objet de type ArrayList.
- Créer une classe Machine, chaque instance de cette classe contiendra les informations listées ci-dessus pour un ordinateur. Chaque instance sera stockée dans l'objet ArrayList.

B) La classe StringCollection

C'est un cas particulier de tableau dynamique. Un objet de type StringCollection peut contenir 0, une ou plusieurs chaînes de caractères. Il peut contenir plusieurs fois la même chaîne.

On y retrouve la propriété count et les méthodes Add, Insert, Remove, Clear, Contains, etc.

C) La classe Stack

La classe Stack implémente une pile de type LIFO (Last In First Out). Cette pile contient différents objets, mais l'accès est plus limité que l'accès à un tableau. La pile s'agrandit automatiquement en fonction des dépôts d'objets.

Classe Stack		
Stack ← Object		
using System.Collections;		
Propriété de la classe Stack		
Count	int	Nombre d'éléments dans la pile.
Méthodes de la classe Stack		
Stack();		Constructeur.
void Clear();		Vide la pile de son contenu.
void Push(Object obj);		Dépose un nouvel objet sur la pile.
object Pop();		Renvoie, en le retirant, l'objet qui se trouve au sommet de la pile. L'exception InvalidOperationException est générée si la pile est vide.
object Peek();		Renvoie, sans le retirer, l'objet qui se trouve au sommet de la pile.
void CopyTo(Array, int index);		Copie le contenu de la pile dans le tableau de taille fixe passé en argument, à partir de la index-ième position dans le tableau.
IEnumerator GetEnumerator();		Renvoie un énumérateur pour le balayage de la pile. Même si l'accès à une pile se fait habituellement par les fonctions Push et Pop, rien n'empêche de balayer toute la pile à l'aide d'un foreach ou des fonctions de Ienumerator. Balayer une pile à l'aide d'un for est impossible car l'opérateur [] ne peut être appliqué à une pile.
object[] ToArray();		Renvoie un tableau contenant tous les éléments de la pile.

Nous dépassons ici deux options sur la pile :

D) La classe Queue C'est une pile de type FIFO (First In, First Out).

Classe Stack		
Stack ← Object		
using System.Collections;		
Propriété de la classe Stack		
Count	int	Nombre d'éléments dans la pile.
Méthodes de la classe Stack		
Stack();		Constructeur.
void Clear();		Vide la pile de son contenu.
void Push(Object obj);		Dépose un nouvel objet sur la pile.

Méthodes de la classe Queue (*suite*)

Queue();	Constructeur.
void Clear();	Vide la file de son contenu.
void Enqueue(Object obj);	Ajoute un nouvel objet.
object Dequeue();	Renvoie, en le retirant, l'objet qui se trouve en tête de file. L'exception InvalidOperationException est générée si la file est vide.
object Peek();	Renvoie, sans le retirer, l'objet qui se trouve au sommet de la file.

E) Les listes triées.

La classe SortedList implémente une liste triée selon une clé et sans possibilité de doublons, ce qui implique que toutes les clés doivent être différentes. Il faut faire la différence entre la clé (par exemple l'identificateur d'une personne) et la valeur associée à cette clé (par exemple nom, prénom de la personne). Clé et valeur sont des objets. L'ensemble est trié selon la clé et l'opérateur [] donne accès à une valeur particulière sur base d'une clé.

Classe SortedList

```
SortedList ← Object
```

```
using System.Collections;
```

Propriétés de la classe SortedList

Capacity	int	Nombre d'éléments que peut contenir la liste. Cette propriété présente peu d'intérêt puisque l'accroissement de taille est automatique.
Count	int	Nombre d'éléments présents dans la liste.
Item		Indexeur d'une liste triée. L'indexeur, entre crochets, doit être un objet « clé ».
Keys	coll	Collection des clés (voir exemples).
Values	coll	Collection des valeurs.

Méthodes de la classe SortedList

SortedList();	Constructeur. Initialement, la capacité de la liste est de seize valeurs. Cette capacité est automatiquement augmentée quand cela s'avère nécessaire.
SortedList(int);	Même chose mais en spécifiant une capacité initiale, ce qui peut améliorer les performances puisque toute réorganisation prend du temps.
SortedList(IComparer);	Constructeur avec spécification d'une classe de comparaison. Cela n'est nécessaire que si la classe des objets de la liste triée n'implémente pas l'interface IComparable.
int Add(object key, object obj);	Ajoute un couple (clé/valeur). L'exception ArgumentException est générée si la clé est déjà présente.
void Clear();	Vide la liste triée de ses éléments.
bool Contains(object key);	Renvoie true si la liste contient la clef key.

bool ContainsKey(object key);	Semblable à la précédente.
bool ContainsValue(object val);	Renvoie true si la liste contient la valeur val.
object GetByIndex(int index);	Renvoie l'objet en index-ième position.
IDictionaryEnumerator GetEnumerator();	Renvoie un énumérateur pour le balayage de la liste triée. Les méthodes Move-Next et Reset de l'énumérateur peuvent être utilisées. La propriété et les deux méthodes d'IEnumerator ont été présentées lors de l'étude de la classe ArrayList. Les propriétés Key et Value appliquées à l'énumérateur fournissent respectivement la clé et la valeur (voir exemple ci-dessous).
object GetKey(int index);	Renvoie l'objet en index-ième position.
IList GetKeyList();	Renvoie une liste de clés.
IList GetValueList();	Renvoie une liste de valeurs (voir exemple plus loin).
int IndexOfKey(object key);	Renvoie l'index (dans le tableau des clés) de la clé key.
int IndexOfValue(object value);	Renvoie l'index (dans le tableau des valeurs) de la valeur value.
void Remove(object key);	Supprime la clé (et la valeur) passée en argument.
void RemoveAt(int pos);	Retire l'objet en pos-ième position.
void RemoveRange(int pos, int nb);	Retire nb objets à partir de la pos-ième position.
void TrimToSize();	Ajuste la taille de la liste au nombre d'éléments présents.

- Exemple

```
using System;
using System.Collections;
public class SamplesSortedList {
    public static void Main() {
        // Creates and initializes a new SortedList.
        SortedList mySL = new SortedList();
        mySL.Add("First", "Hello");
        mySL.Add("Second", "World");
        mySL.Add("Third", "!");
        // Displays the properties and values of the SortedList.
        Console.WriteLine("mySL");
        Console.WriteLine("Count: {0}", mySL.Count);
        Console.WriteLine("Capacity: {0}", mySL.Capacity);
        Console.WriteLine("Keys and Values:");
        PrintKeysAndValues(mySL);
    }

    public static void PrintKeysAndValues(SortedList myList) {
        Console.WriteLine("\t-KEY-\t-VALUE-");
        for (int i = 0; i < myList.Count; i++) {
            Console.WriteLine("\t{0}:\t{1}", myList.GetKey(i), myList.GetByIndex(i));
        }
        Console.WriteLine();
    }
}

/*
This code produces the following output.

mySL
Count: 3
Capacity: 16
Keys and Values:
-KEY- -VALUE-
First: Hello
Second: World
Third: !
*/
```

F) La classe Hashtable.

La classe Hashtable implémente un conteneur appelé "bag" (panier), ou dictionnaire. Les objets sont insérés sans ordre particulier, et sont associés à une clé unique. Les objets sont rangés dans différents compartiments sur la base de la clé. Un compartiment peut contenir plusieurs objets. Lors d'un accès au panier, la fonction d'accès appelle une fonction de hachage, GetHashCode qui renvoie un nombre sur base d'une clé. Ce nombre donne directement accès à un compartiment qui comprend 0 ou plusieurs objets. La recherche devient alors séquentielle dans le compartiment.

En fait si la fonction de hachage renvoie toujours la même valeur, tous les objets sont rangés dans le même compartiment, et la recherche séquentielle dans ce compartiment va ralentir les accès au panier.

Les propriétés et méthodes de Hashtable sont semblables à celles de SortedList, sauf que le panier n'est pas trié. La clé doit être unique.

- **La classe StringDictionary**

Elle a les mêmes caractéristiques que Hasthtable, mais la clé et l'objet à insérer doivent être des chaînes de caractères.

- **La classe NameValueCollection**

Elle est semblable à StringDictionary, mais la clé ne doit pas être unique. L'accès à la collection peut être effectué par un index, celui-ci pouvant être de type string (la clé elle-même) ou de type entier (numéro d'ordre dans la collection).

G) Les tableaux de bits.

Un objet de type BitArray peut contenir des éléments dont la valeur est true ou false.

Cet objet peut être indexé par [] pour déterminer ou modifier un bit particulier.

Constructeurs, méthodes et propriétés: voir page suivante.

Classe BitArray

```
BitArray ← Object  
using System.Collections;
```

Propriétés de la classe BitArray

Count	int	Nombre d'éléments présents dans le tableau de bits.
Item		Indexeur d'un tableau de bits.

Constructeurs de la classe BitArray

BitArray(int);	L'argument indique le nombre de bits dans le tableau. Ils sont tous initialisés à false.
BitArray(int, bool);	Le premier argument indique le nombre de bits et le second leur valeur initiale (true ou false).
BitArray(bool[]);	Initialisation du tableau à partir d'un tableau de booléens.
BitArray(byte[]);	Initialisation du tableau à partir d'un tableau d'octets.

Méthodes de la classe BitArray

BitArray And(BitArray val);	Effectue un AND entre chaque bit du tableau sur lequel porte l'opération et les bits correspondant du tableau passé en argument.
bool Get(int n);	Renvoie la valeur du bit en <i>n-ième</i> position (le bit 0 correspond à celui d'extrême gauche). L'opérateur [] remplace avantageusement Get.
BitArray Not();	Inverse chaque bit du tableau : true devient false et false devient true.
BitArray Or(BitArray val);	Comme AND mais un OU est effectué.
void Set(int n, bool val);	Modifie la valeur du <i>n-ième</i> bit. L'opérateur [] remplace avantageusement Set.
void SetAll(bool val);	Fait passer tous les bits du tableau à la valeur passée en argument.
BitArray Xor(BitArray val);	Comme AND mais un XOR est effectué.

- **Exemple**

Voir page suivante

```
using System;
using System.Collections;
public class SamplesBitArray {

    public static void Main() {
        // Creates and initializes several BitArrays.
        BitArray myBA1 = new BitArray( 5 );

        BitArray myBA2 = new BitArray( 5, false );

        byte[] myBytes = new byte[5] { 1, 2, 3, 4, 5 };
        BitArray myBA3 = new BitArray( myBytes );

        bool[] myBools = new bool[5] { true, false, true, true, false };
        BitArray myBA4 = new BitArray( myBools );

        int[] myInts = new int[5] { 6, 7, 8, 9, 10 };
        BitArray myBA5 = new BitArray( myInts );

        // Displays the properties and values of the BitArrays.
        Console.WriteLine( "myBA1" );
        Console.WriteLine( "    Count:      {0}", myBA1.Count );
        Console.WriteLine( "    Length:     {0}", myBA1.Length );
        Console.WriteLine( "    Values:" );
        PrintValues( myBA1, 8 );

        Console.WriteLine( "myBA2" );
        Console.WriteLine( "    Count:      {0}", myBA2.Count );
        Console.WriteLine( "    Length:     {0}", myBA2.Length );
        Console.WriteLine( "    Values:" );
        PrintValues( myBA2, 8 );

        Console.WriteLine( "myBA3" );
        Console.WriteLine( "    Count:      {0}", myBA3.Count );
        Console.WriteLine( "    Length:     {0}", myBA3.Length );
        Console.WriteLine( "    Values:" );
        PrintValues( myBA3, 8 );

        Console.WriteLine( "myBA4" );
        Console.WriteLine( "    Count:      {0}", myBA4.Count );
        Console.WriteLine( "    Length:     {0}", myBA4.Length );
        Console.WriteLine( "    Values:" );
        PrintValues( myBA4, 8 );

        Console.WriteLine( "myBA5" );
        Console.WriteLine( "    Count:      {0}", myBA5.Count );
        Console.WriteLine( "    Length:     {0}", myBA5.Length );
        Console.WriteLine( "    Values:" );
        PrintValues( myBA5, 8 );
    }

    public static void PrintValues( IEnumerable myList, int myWidth ) {
        int i = myWidth;
        foreach ( Object obj in myList ) {
            if ( i <= 0 ) {
                i = myWidth;
                Console.WriteLine();
            }
            i--;
            Console.Write( "{0,8}", obj );
        }
        Console.WriteLine();
    }
}
```