

# Utilisation de la classe ArrayList

(J-C Armici [jca.developpez.com](http://jca.developpez.com), [www.unvrai.com](http://www.unvrai.com))

## Remarque préliminaire:

*Ce qui suit a été réalisé avec Visual Studio 2005 professionnel (en anglais, car la version française n'est pas encore disponible)*

## Objectif:

Le namespace **System.Collection** du framework .NET définit un certain nombre de classes gérant des collections, également appelées classes conteneurs. Parmi ces classes permettant de gérer des objets se présentant comme des collections on peut citer:

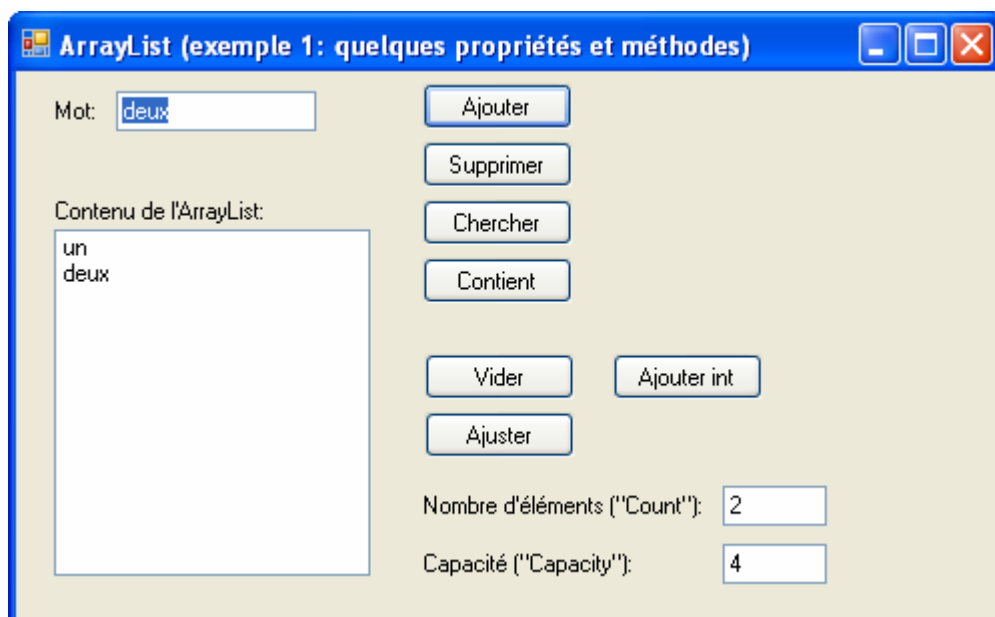
- ArrayList
- HashTable
- BitArray
- SortedList
- Queue
- Stack

Le but de ce document est d'illustrer l'utilisation de la classe ArrayList au travers de quelques exemples. Nous pourrions également comparer les tableaux et les collections.

## Exemple 1: ArrayList

Afin de mieux comprendre comment utiliser un ArrayList nous allons créer un exemple d'application permettant d'illustrer quelques unes des propriétés et méthodes de la classe ArrayList. Il ne s'agit pas d'une présentation exhaustive; pour cela il suffit de consulter, par exemple, la librairie MSDN.

Voici comment se présente notre exemple:



Ce programme intègre les possibilités suivantes:

- adjonction d'un élément (chaîne de caractères) dans un ArrayList
- suppression d'un élément dans un ArrayList
- recherche d'un élément dans un ArrayList
- test si l'ArrayList contient un élément donné
- suppression de tous les éléments d'un ArrayList
- visualisation du nombre d'éléments ainsi que de la capacité d'un ArrayList

Notre **ArrayList** s'appelle **Liste** et est déclaré de la manière suivante:

```
ArrayList Liste = new ArrayList();
```

Il convient de signaler qu'à aucun moment il est spécifié le type des éléments placés dans la liste, puisqu'un **ArrayList** est justement une liste d'objets.

Afin de visualiser le contenu de notre ArrayList au fur et à mesure des opérations nous avons utilisé un **ListBox**. Cette visualisation est obtenue en appelant la méthode **MontrerListe()**:

```
private void MontrerListe()
{
    lbListe.Items.Clear();
    foreach (object mot in Liste)           // parcours de l'ArrayList
        lbListe.Items.Add(mot);
    txtCount.Text = Liste.Count.ToString(); // affichage du nombre d'éléments
    txtCapacity.Text = Liste.Capacity.ToString(); // affichage de la capacité de l'ArrayList
}
```

On remarquera l'utilisation très pratique de la boucle **foreach** permettant d'itérer sur tous les éléments d'un ensemble énumérable. La méthode **MontrerListe()** affiche également le nombre d'éléments contenus dans l'**ArrayList** (**Liste.Count**), ainsi que sa capacité actuelle (**Liste.Capacity**).

### Adjonction d'un élément:

Voici le code exécuté pour ajouter un élément:

```
private void btnAjouter_Click(object sender, EventArgs e)
{
    Liste.Add(txtMot.Text);           // ajout du mot
    MontrerListe();                   // affichage de l'ArrayList

    // juste un peu d'ergonomie
    ActiveControl = txtMot;
    txtMot.SelectionStart = 0;
    txtMot.SelectionLength = txtMot.Text.Length;
}
```

Les trois dernières lignes servent uniquement à faciliter la saisie et l'adjonction en continu de mots dans l'ArrayList. L'adjonction proprement dite est effectuée par la méthode **Add**.

### Suppression d'un élément:

La méthode **Remove** supprime l'élément spécifié. Si le même élément se trouve plusieurs fois dans la liste, la première occurrence trouvée est supprimée.

```
private void btnSupprimer_Click(object sender, EventArgs e)
{
    Liste.Remove(txtMot.Text);
    MontrerListe();
}

```

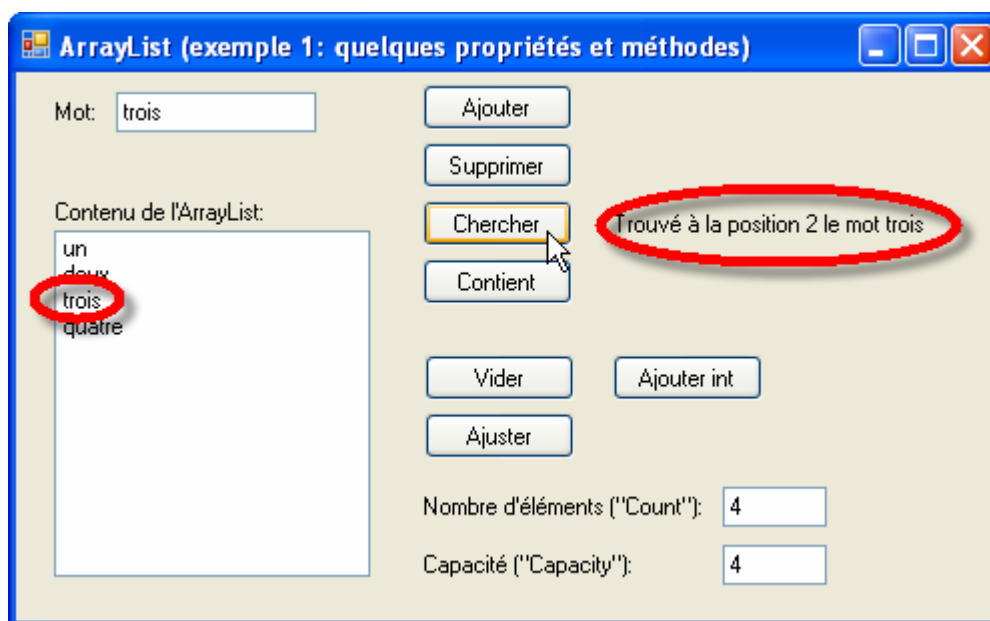
### Recherche d'un élément:

En cliquant sur le bouton **Chercher** le programme indique si l'élément spécifié est dans la liste et à quelle position. La méthode **BinarySearch** peut être utilisée pour effectuer une telle recherche. Si l'élément n'est pas trouvé une valeur négative est retournée. Si l'élément est trouvé, sa position est retournée:

```
private void btnChercher_Click(object sender, EventArgs e)
{
    int i;
    i = Liste.BinarySearch(txtMot.Text);
    if (i < 0)
        lTrouve.Text = "Elément non trouvé";
    else
        lTrouve.Text = "Trouvé à la position " + i.ToString() + " le mot " + Liste[i];
}

```

Voici un exemple:



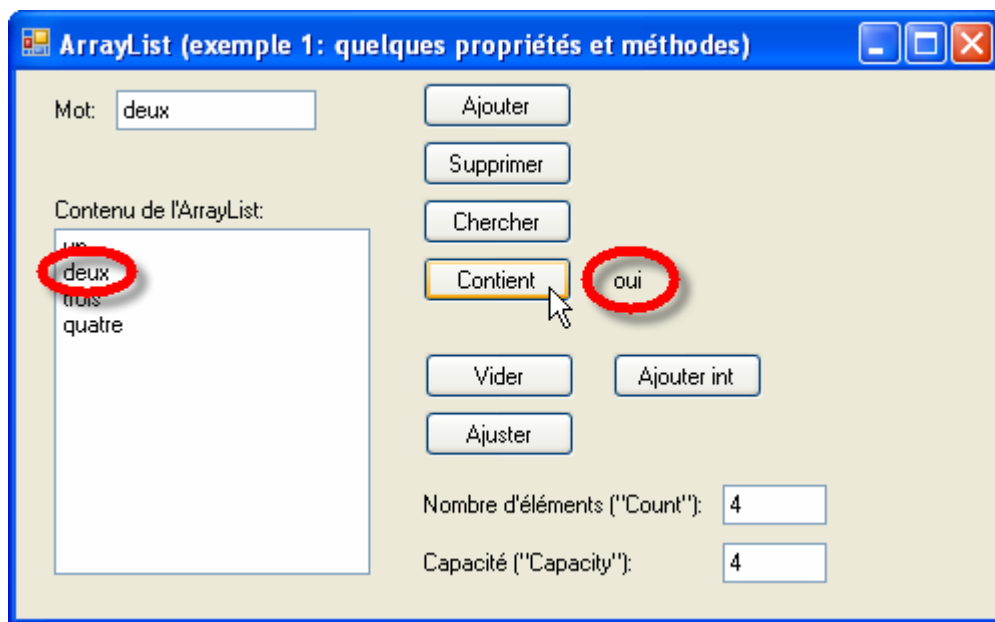
### Test d'appartenance:

Il peut parfois être utile de savoir si un élément se trouve dans un ArrayList, sans forcément avoir besoin de connaître sa position. Dans ce cas, on peut faire appel à la méthode **Contains()** qui retourne une valeur booléenne indiquant si l'élément se trouve dans l'ArrayList.

Voici le code correspondant:

```
private void btnContient_Click(object sender, EventArgs e)
{
    if (Liste.Contains(txtMot.Text))
        lContient.Text = "oui";
    else
        lContient.Text = "non";
}
```

Et voici un exemple:



### Suppression du contenu:

Voici le code exécuté sur le clic du bouton **Vider**:

```
private void btnClear_Click(object sender, EventArgs e)
{
    Liste.Clear();
    MontrerListe();
}
```

C'est donc la méthode **Clear** qui supprime tous les éléments d'un ArrayList.

Il faut préciser que la méthode **Clear** ne modifie pas la capacité de l'ArrayList

### Ajuster la capacité

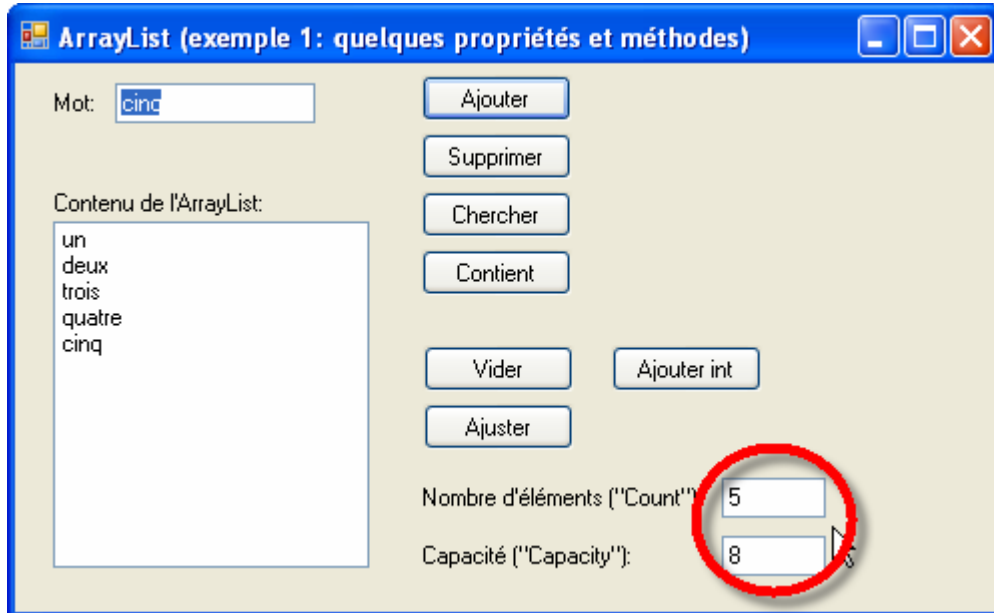
Il peut être utile d'ajuster la capacité d'un ArrayList au nombre d'éléments effectivement contenus. Cela conduit à expliquer de quelle manière la capacité d'un ArrayList est automatiquement augmentée au fur et à mesure des besoins.

Par défaut la capacité est de 4 au moment de la création d'un ArrayList. Dès que 4 éléments y sont ajoutés et que l'on ajoute un 5<sup>ème</sup> élément, la capacité passe à 8. Lors de l'ajout d'un 9<sup>ème</sup> élément, elle passe à 16, et ainsi de suite. La capacité est doublée chaque fois que le nombre d'éléments nécessite une augmentation de la capacité.

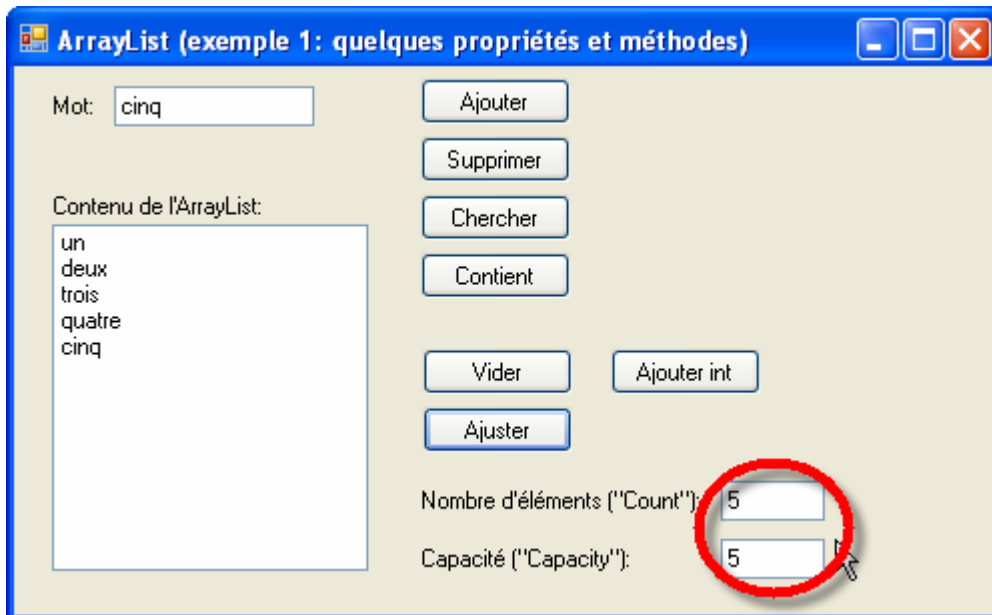
Le bouton **Ajuster** appelle le code suivant:

```
private void btnAjuster_Click(object sender, EventArgs e)
{
    Liste.TrimToSize();
    MontrerListe();
}
```

La méthode **TrimToSize** ramène la capacité au nombre effectif d'éléments dans l'ArrayList. Dans le cas suivant:



l'ajustement de la capacité (clic sur le bouton **Ajuster**) conduit à:



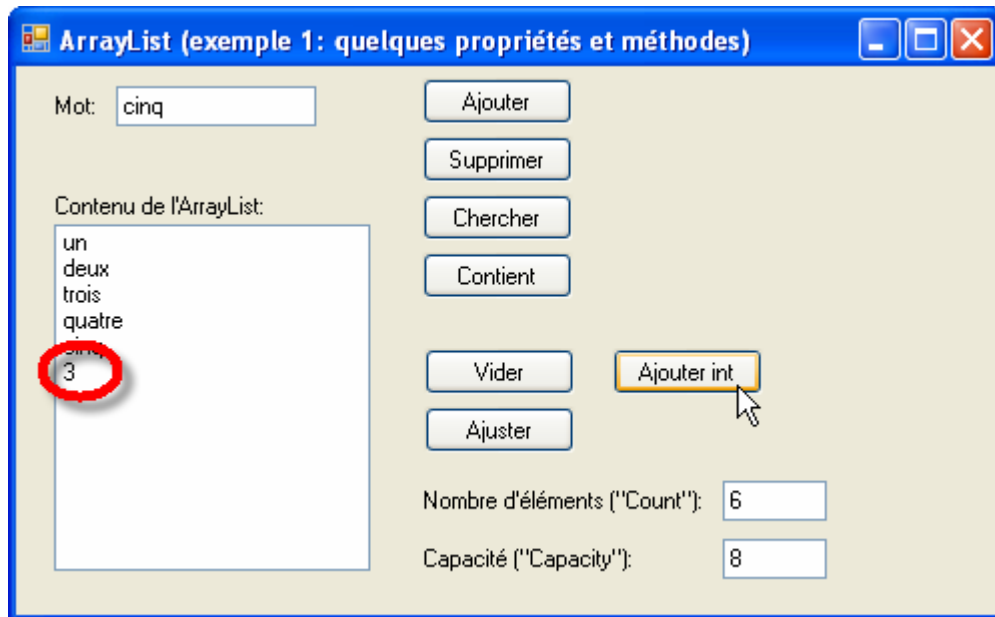
En partant de cette situation, l'ajout d'un nouvel élément ne fait pas passer la capacité à 8, mais à 10. L'augmentation automatique ne correspond donc pas à des puissances de deux, mais bien à un doublement de la capacité actuelle. Dans notre exemple, la capacité passerait à 10, 20, 40, etc.

## Adjonction d'un nombre:

Le bouton **Ajouter int** exécute le code suivant:

```
private void btnAddInt_Click (object sender, EventArgs e)
{
    Liste.Add(3);
    MontrerListe ();
}
```

Comme on peut le voir, le nombre entier 3 est ajouté à l'ArrayList qui, jusqu'ici, contenait des chaînes de caractères.



Le parcours des éléments de l'ArrayList effectué dans la méthode MontrerListe qui permet de remplir le ListBox de visualisation est écrit sous la forme:

```
foreach (object mot in Liste) // parcours de l'ArrayList
    lbListe.Items.Add(mot);
```

La boucle parcourt des objets et non des chaînes de caractères (string). C'est la raison pour laquelle lors de l'ajout d'un entier la boucle s'exécute sans problème. Si, en revanche on avait écrit cette boucle sous la forme

```
foreach (string mot in Liste) // parcours de l'ArrayList
    lbListe.Items.Add(mot);
```

l'exécution du programme aurait déclenché une exception lors de l'ajout d'un nombre entier.

## Accès aux éléments

On peut accéder aux éléments d'un ArrayList en utilisant la même notation que pour les tableaux. Par exemple, l'instruction suivante:

```
Liste[2] = "rien";
```

Place le mot "rien" à la position 2 de l'ArrayList. Dans notre exemple ci-dessus le mot "trois" est remplacé par le mot "rien".

## Tableaux et collections

Voici en quoi les tableaux ordinaires diffèrent des collections (par exemple ArrayList):

- la déclaration d'un tableau spécifie le type des éléments qu'il va contenir. Ceci n'est pas le cas d'une collection, car les éléments d'une collection sont des objets
- un tableau a une taille fixe et déterminée, alors que la taille d'une collection peut être changée dynamiquement
- il est possible d'utiliser une collection en lecture seule (méthode ReadOnly)

Dans bien des cas l'utilisation d'un ArrayList en lieu et place d'un tableau se révèle avantageuse.

### Exemple 2: création d'un tableau en partant d'un ArrayList

Dans ce deuxième exemple nous allons créer un ArrayList contenant des objets de type **Personne**. La classe **Personne** (très simplifiée) est définie comme suit

```
using System;
using System.Collections.Generic;
using System.Text;

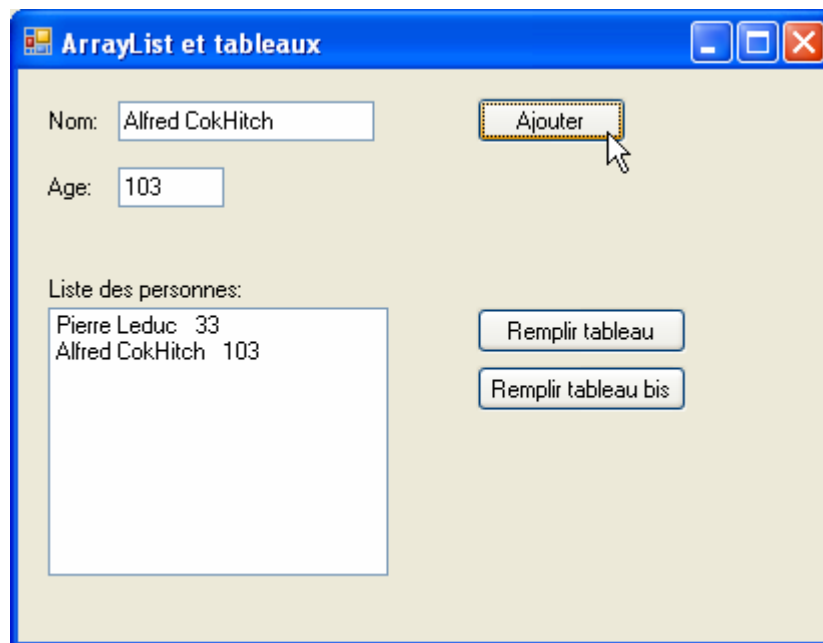
namespace ArrayListTableau
{
    class Personne
    {
        string _nom;
        int _age;

        #region Propriétés
        public string Nom
        {
            get { return _nom; }
            set { _nom = value; }
        }
        public int Age
        {
            get { return _age; }
            set { _age = value; }
        }
        #endregion

        public Personne():this("",0)
        {
        }

        public Personne(string LeNom, int LAge)
        {
            Nom = LeNom;
            Age = LAge;
        }
    }
}
```

Pour travailler avec cette classe `Personne` nous utilisons une application Winform ayant l'aspect suivant:



Notre `ArrayList ListeP` est déclaré de la même manière que dans l'exemple précédente:

```
ArrayList ListeP = new ArrayList();
```

Le bouton **Ajouter** exécute le code suivant:

```
private void btnAjouter_Click(object sender, EventArgs e)
{
    Personne p = new Personne(txtNom.Text, int.Parse(txtAge.Text)); // nouvelle personne
    ListeP.Add(p); // ajout de la personne à la liste
    MontrerListe(); // affichage de la liste
}
```

L'adjonction ne présente aucune différence avec l'exemple précédent et la méthode **MontrerListe** peut s'écrire de la manière suivante:

```
private void MontrerListe()
{
    lbPersonnes.Items.Clear();
    foreach (Personne p in ListeP)
        lbPersonnes.Items.Add(p.Nom + " " + p.Age);
}
```

Mais ce qui nous intéresse ici c'est comment transférer le contenu de notre `ArrayList` dans un tableau. Nous allons voir deux possibilités, correspondant aux boutons **Remplir tableau** et **Remplir tableau bis**.

La méthode qui permet d'obtenir un tableau est **ToArray**. Cette méthode peut fournir un tableau générique d'objets, ou bien un tableau d'objets d'un type spécifié. Dans le premier cas voici comment on peut procéder:

```

private void btnRemplirT_Click(object sender, EventArgs e)
{
    // version avec tableau de type Object
    Object[] TabP = ListeP.ToArray();

    // affichage dans le ListBox
    lbPersonnes.Items.Clear();
    foreach (object ob in TabP)
        lbPersonnes.Items.Add("--> " + ((Personne)ob).Nom + " " + ((Personne)ob).Age);
}

```

Une fois le tableau **TabP** rempli avec les éléments de l'ArrayList une boucle affiche les éléments du tableau dans le **ListBox**. Comme on peut le voir il faut faire appel à un cast pour afficher les champs de chaque personne.

A noter que nous aurions également pu écrire la boucle sous la forme for plutôt que foreach:

```

for (int i = 0; i < TabP.Length; i++)
    lbPersonnes.Items.Add("--> " + ((Personne)TabP[i]).Nom + " " + ((Personne)TabP[i]).Age);

```

Cette écriture est un peu lourde à manipuler. Nous pouvons simplifier l'accès aux champs des personnes en faisant appel à la deuxième forme proposée pour la méthode **ToArray**. C'est le code exécuté par le bouton **Remplir tableau bis**:

```

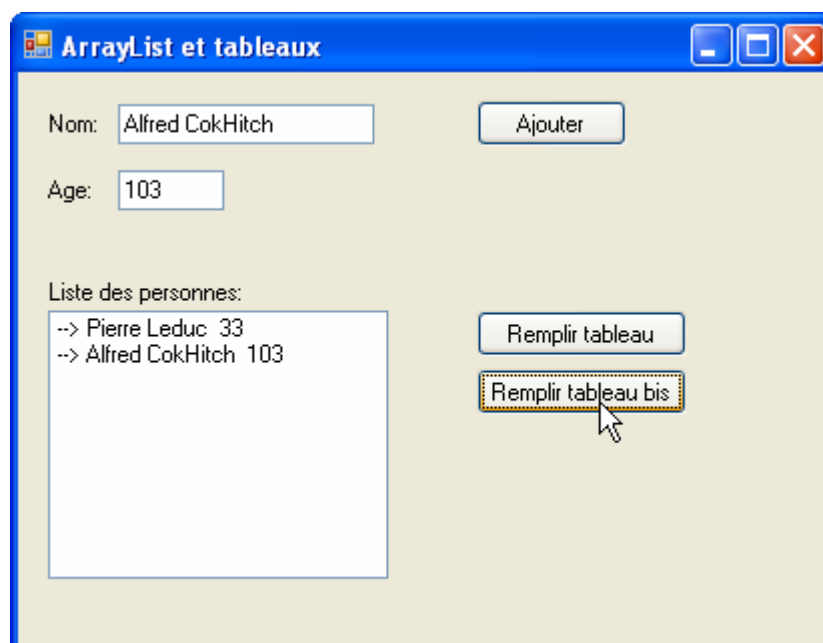
private void bntRemplirT2_Click (object sender, EventArgs e)
{
    // version avec tableau de type Personne (utilisation moins lourde)
    Personne[] TabP = (Personne[])ListeP.ToArray (typeof(Personne));

    // affichage dans le ListBox
    lbPersonnes.Items.Clear();
    foreach (Personne p in TabP)
        lbPersonnes.Items.Add("--> " + p.Nom + " " + p.Age);
}

```

Le type **Personne** est spécifié à la méthode **ToArray**, ce qui permet de travailler ensuite directement avec un tableau de type **Personne**.

L'effet obtenu en cliquant sur les deux boutons est le même:

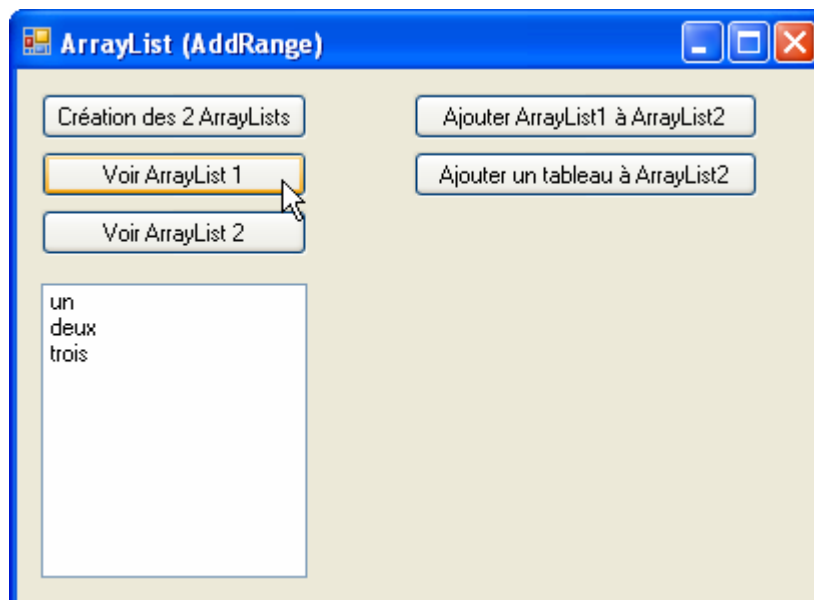


### Exemple 3: adjonction en bloc d'éléments dans un ArrayList

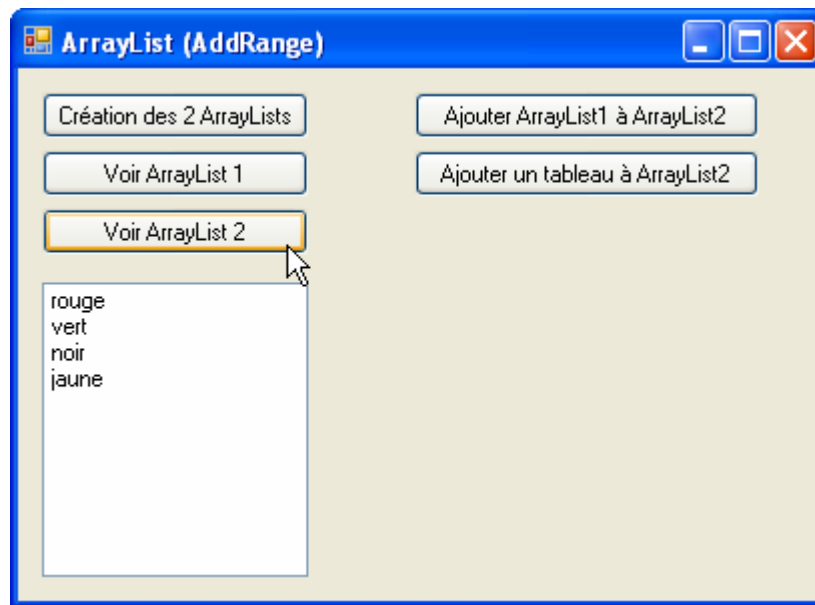
Cet exemple illustre l'utilisation de la méthode **AddRange** de la classe ArrayList. Le programme se présente sous la forme suivante:



Le seul bouton disponible crée deux ArrayList, **a1** et **a2**. Une fois créés, on peut les visualiser à l'aide des deux boutons correspondants. Voici le contenu de **a1**:



et le contenu de **a2**:



Le code de création des ArrayList est banal:

```
private void btnCreer_Click(object sender, EventArgs e)
{
    a1 = new ArrayList();
    a2 = new ArrayList();

    a1.Add("un");
    a1.Add("deux");
    a1.Add("trois");

    a2.Add("rouge");
    a2.Add("vert");
    a2.Add("noir");
    a2.Add("jaune");

    // activation des boutons de visualisation
    btnVoir1.Enabled = true;
    btnVoir2.Enabled = true;
    btnAjouter.Enabled = true;
    btnAjouter2.Enabled = true;
}
```

La partie du code permettant de visualiser l'un ou l'autre ArrayList est le suivant:

```
private void VoirListe(ArrayList liste)
{
    lbListe.Items.Clear();
    foreach (object ob in liste)
        lbListe.Items.Add(ob);
}

private void btnVoir1_Click(object sender, EventArgs e)
{
    VoirListe(a1);
}

private void btnVoir2_Click(object sender, EventArgs e)
{
    VoirListe(a2);
}
```

On peut alors ajouter à **a2** le contenu de **a1**:

```
private void btnAjouter_Click(object sender, EventArgs e)
{
    a2.AddRange(a1);
}
```

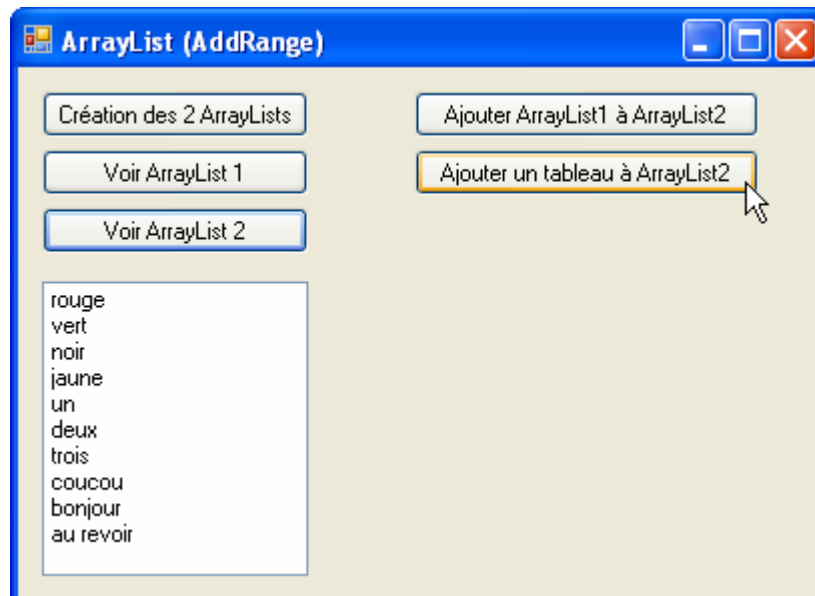
et le résultat, si on affiche le contenu de **a2**, est:



Il est également possible d'ajouter, par exemple, le contenu d'un tableau dans **a2**:

```
private void btnAjouter2_Click(object sender, EventArgs e)
{
    string[] tab = { "coucou", "bonjour", "au revoir" };
    a2.AddRange(tab);
}
```

On obtient alors:



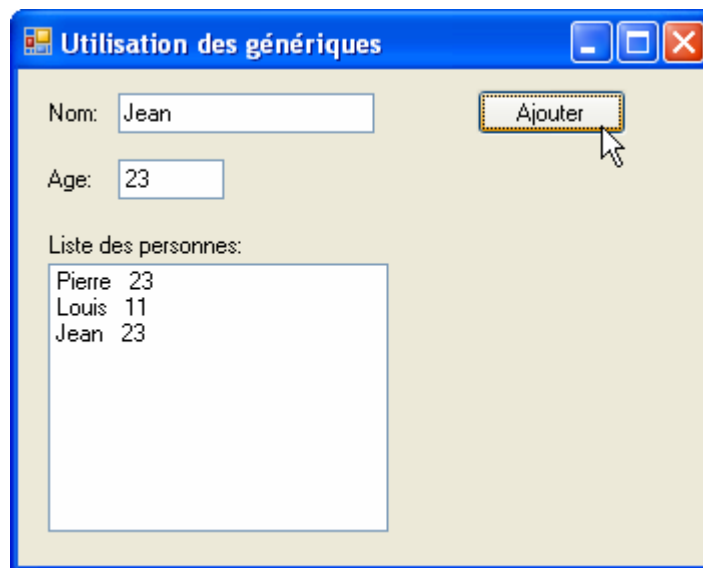
## Exemple 4: utilisation des génériques

Pour terminer nous allons reprendre l'exemple 2 et le traiter de manière plus moderne et innovante.

Dans C# 2.0 a été introduite une nouveauté très intéressante: les génériques. Nous n'allons pas ici faire une description complète des génériques. Dans le but de susciter de l'intérêt pour ce nouveau concept, nous allons uniquement montrer comment notre deuxième exemple peut être traité avec les génériques. L'évidente simplicité du code devrait pousser chacun à utiliser les génériques.

Cet exemple utilise la même classe **Personne** que l'exemple 2.

Voici l'aspect du programme:



La déclaration de notre liste générique est :

```
List<Personne> pers = new List<Personne>();
```

On notera la nouvelle écriture: le type des éléments est compris entre < >.

Voici le reste du programme, à savoir l'affichage des éléments dans le ListBox et l'ajout d'une personne:

```
private void MontrerListe()
{
    lbPersonnes.Items.Clear();
    foreach (Personne p in pers)
        lbPersonnes.Items.Add(p.Nom + " " + p.Age);
}

private void btnAjouter_Click(object sender, EventArgs e)
{
    pers.Add( new Personne(txtNom.Text, int.Parse(txtAge.Text))); // nouvelle personne
    MontrerListe();
}
```

Comme on peut le voir, le code devient désarmant de simplicité. Il n'y a plus besoin d'aucun cast.