**1. Les listes**

**1.1 Gestion des éléments d'une liste**

* La liste est une alternative aux tableaux: la **taille** d'un tableau est fixe - celle **d'une liste peut varier**
* On parle **d'index** de la liste et non pas d'indice
* On peut ajouter, retirer des éléments de la liste sans altérer les autres éléments.
* son avantage : elle **n'a pas de dimension fixe**, on peut augmenter ou diminuer le nombre de ses éléments
* Syntaxe pour créer une collection List <T> est comme suit

**List <string>** *lesJours* **= new List<string>();**

**List <type>** *nomListe* **= new List <type> ();**

Méthodes associées à la gestion des listes

|  |  |
| --- | --- |
| Ajout d'élément | *nomListe*.**Add**(valeur) |
| Suppression en connaissant la valeur de l'élément | *nomListe*.**Remove**(valeur) |
| Suppression en connaissant le rang de l'élément | *nomListe*.**RemoveAt**(rang) |
| Insertion d'un nouvel élément | *nomListe*.**Insert**(rang, valeur) |
| Recherche de la valeur d'un élément en connaissant son rang | *nomListe*[rang] |
| Recherche du rang d'un élément dont on sait la valeur | *nomListe*.**IndexOf**(valeur) |
| Changement de valeur d'un élément | *nomListe*[rang] = newValeur |
| Compter les éléments de la liste | *nomListe*.**Count** |
| Trier les éléments d'une liste | *nomListe*.**Sort()** |
| Balayage des éléments d'une liste | **foreach** (type *cetElement*  in *nomListe*)  { Action sur ( *cetElement* ); } |

**1.2 Un exemple de gestion d'une liste de couleurs**

static void Main()

{ *// Création et affectation d'éléments à la liste des couleurs*

**List<string> colorList = new List<string>();**

colorList.Add ("Rouge");

colorList.Add ("Vert");

colorList.Add ("Jaune");

colorList.Add ("Orange");

*// Affichage du contenu intitialisé de la liste*

**foreach (string color in colorList)**

{ Console.WriteLine(color); }

*// Visualisation de la couleur de rang 1 puis modification, c'est à dire le deuxième élément de la liste*

**Console.WriteLine(("Affichage de la couleur au rang 1 : " + colorList[1]);**

**colorList[1] = "Indigo";**

**Console.WriteLine("Modification de la couleur de rang 1 par : " + colorList[1]);**

*// Suppression puis insertion d'un nouvel élément*

Console.WriteLine("Suppression de l'élément de valeur orange");

**colorList.Remove("Orange");**

Console.WriteLine("Insertion de l'élément blanc au rang 2");

**colorList.Insert(2, "Blanc");**

**foreach (string color in colorList)**

{ Console.WriteLine(color); }

*// On utilise IndexOf*

Console.WriteLine("Index de l'élément de valeur Jaune : ");

**Console.WriteLine(colorList.IndexOf("Jaune"));**

**colorList[colorList.IndexOf("Jaune")] = "Noir";**

**foreach (string color in colorList)**

{ Console.WriteLine(color); }

*// initialisation des éléments de la liste*

**colorList.Clear();**

} }



**1.3 Liste d'employés**

Vous devez produire un programme qui permette de gérer une liste dynamique d'employés.

La liste est la suivante :

Poivre Romain

Safran Pauline

Cannelle Coline

Coriandre Joël

La liste étant créée, on veut afficher les résultats : rang de chaque élément et contenu

Puis on ajoute un nouvel élément en tête de liste : Du Sel Marine

On affiche le nombre d'éléments de la liste

Enfin on affiche la liste alphabétique des employés

**A faire :** Ecrire le programme C# permettant de réaliser ces objectifs de gestion de liste

static void Main(string[] args)

{

// Création et affectation d'éléments à la liste des employés

**List<string> listEmployes = new List<string>();**

**listEmployes.Add ("Poivre Romain ");**

**listEmployes.Add ("Safran Pauline ");**

**listEmployes.Add ("Cannelle Coline ");**

**listEmployes.Add ("Coriandre Joël");**

// Affichage du contenu initialisé de la liste des employés

Console.WriteLine("Liste initiale des employés");

**foreach (string unEmploye in listEmployes)**

**{**

**Console.WriteLine("rang : " + listEmployes.IndexOf(unEmploye) + " nom de l'employé : " + unEmploye);**

**}**

// insertion d'un nouvel employé

**listEmployes.Insert(0, "Du Sel Marine ");**

Console.WriteLine();

Console.WriteLine("Affichage de la liste après insertion de Du Sel marine");

foreach (string unEmploye in listEmployes)

{

Console.WriteLine(unEmploye);

}

// comptage des éléments de la liste

Console.WriteLine();

**Console.WriteLine("La liste comporte {0} éléments ", listEmployes.Count());**

// tri alphabétique de la lsite des employés

**listEmployes.Sort();**

Console.WriteLine();

Console.WriteLine("Affichage de la liste après tri");

}

}

}



**2. Classe générique**

**2.1 C'est quoi une classe générique ?**

C'est une classe prédéfinie, qui permet d'utiliser des méthodes (ajout, suppression, tri, ...) sans avoir à se soucier du type des données que l'on manipule. Les données peuvent être des entiers, des chaines de caractères, voire même des classes. C'est seulement au moment de la phase de compilation que le type des données est traité.

**Exemple : List<type>** est une classe **générique**

**cette classe permet de gérer des listes de données quel que soit leur type.**

**Exemple** : Les listes peuvent être utilisées avec n'importe quel type : entiers, réels, chaines, ... et même objets

**List**<string> lesEmployes = new List<string>();

List<int> lesEntiers = new List<int>();

List<Voiture> lesVoitures = new List<Voiture>();

List<Secteur> lesSecteurs = new List<Secteur>();

**2.2 List<T> - Queue<T> - Dictionnary<TKey, TValue>**T = type

Il existe en C# plusieurs classes génériques prédéfinies et utilisables dans les programmes :

* les listes LIST **List**<int> *mesEntiers* = **new List**<int>();
* les files QUEUE **Queue**<int> *maFile* = **new Queue**<int>();
* les dictionnaires DICTIONNARY **Dictionnary**<string> *monDico* = **new Dictionnary**<string<();

C'est au moment de la création d'une liste, qu'on déclare le type utilisé pour cette liste particulière.

On a vu que la classe liste<T> est associée à différentes méthodes : **Add, Remove, Count, Sort** ... qui sont utilisables sur les éléments de la liste quel que soit le type de ces éléments.

Une **Queue<T>** est une liste remplie au fur et à mesure et qui est dépilée selon la méthode FIFO

Les méthodes : Pour ajouter un élément : **Enqueue**, pour dépiler : **Dequeue**

Chaque élément ajouté est placé en dernière position avec la méthode Enqueue

Seul le premier élément inséré dans la queue est accessible par la méthode Dequeue

La Queue n'a pas d'index, ni de méthode Remove.

Un **Dictionary<TKey, TValue>** est une collection de paires clef/valeur, chaque clef étant unique :

C'est une classe **doublement générique** : il faut préciser le type des clefs et le type des valeurs.

L'utilisation est proche de celle d'une liste, puisque les deux sont des collections.

Étudions un exemple de dictionnaire faisant l'association entre une extension de fichiers et le programme utilisé par défaut pour ouvrir les fichiers portant cette extension :

**2.3 Foreach : parcours des éléments d'une classe générique**

L'instruction à utiliser est une boucle **foreach**

**foreach (string unEmploye in listEmployes)**

**{**

Console.WriteLine**(unEmploye);**

**}**

**foreach (typeÉlément** *nomÉlément* **in maClasseGénérique)**

**{**

*action réalisée sur chaque élément de la classe générique*

*...........*

**}**

**3. Les collections d'éléments**

* Une **collection est un groupe d'objets de même type**, c'est à dire du type de leur classe d'appartenance.
* <==>Une collection c'est une "liste" d'objets, typés par leur classe d'appartenance.
* Une collection contient un nombre variable d'objets, manipulés avec des méthode (ajout, insertion suppression, ...)
* On peut ajouter, retirer des éléments de la collection sans altérer les autres éléments
* Le parcours des objets de la collection s'effectue avec une instruction **foreach**
* Les collections utilisent la notion de généricité, on dit que ce sont **des types génériques**.

**4. Exercices d'application**

**4.1 Un dictionnaire des extensions d'applications**

On souhaite connaître les noms des applications associées à une extension particulière.

Par exemples :

* on sait qu'un fichier dont le nom est associé à l'extension .docx est un fichier qui sera exécuté à l'aide de l'application Word
* un fichier en extension .pdf sera exécuté avec l'application Adobe Acrobat Reader

On considère ici, qu'à une extension n'est associée qu'une et une seule application

Un programme existant est mis à votre disposition : 4-Dictionnaire\_Application et extension

Les objectifs à atteindre sont les suivants :

* Comprendre comment a été créé le dictionnaire qui associe extension de fichier et application
* Comprendre comment on gère le couple clé, valeur pour insérer un nouvel élément au dictionnaire
* Expliquer comment on gère une erreur par tentative d'ajout d'une extension existante

On peut manipuler chacun des items du dictionnaire les clés **KeyCollection** ou les valeurs **ValueCollection**.

Mais on peut également manipuler l'ensemble clé ET valeur : **KeyValuePair**

**A faire :** Réaliser une trace d'exécution du programme

Ecrire le code qui permet d'associer l'application winword.exe à la clé "rtf"

Ecrire le code qui permet de visualiser l'ensemble des éléments du dictionnaire : extension

(la clé), et application (la valeur)

***Annexe - Le programme C existant***

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

namespace Dico

{ class Program

{ static void Main(string[] args)

{

// Création du dictionnaire : clé = extention; valeur = application

**Dictionary<string, string> mesAppli = new Dictionary<string, string>();**

// Ajout d'éléments. Il ne peut pas y avoir deux clefs identiques mais les valeurs peuvent l'être.

**mesAppli.Add("txt", "notepad++.exe");**

mesAppli.Add("bmp", "paint.exe");

mesAppli.Add("jpg", "paint.exe");

mesAppli.Add("doc", "winword.exe");

mesAppli.Add("rtf", "wordpad.exe");

mesAppli.Add("pdf", "adobe.exe");

// Gestion d'une exception sur la méthode Add : la clef existe déjà.

try

{ mesAppli.Add("txt", "winword.exe"); }

catch (**ArgumentException**)

{ Console.WriteLine("Un élément possède déjà la clef \"txt\"."); }

// Affichage de l'pplication associée à la clé "rtf"

Console.WriteLine("La valeur associée à la clef \"rtf\" est {0}.",mesAppli["rtf"]);

// Si la clef n'existe pas déjà, l'utilisation de l'index crée un nouveau couple clef/valeur.

mesAppli["doc"] = "winword.exe";

// Recherche d'une valeur en fonction de la clé avec gestion d'une exception si la clef n'existe pas

try

{ Console.WriteLine("La valeur associée à la clef \"tif\" est {0}.", mesAppli["tif"]); }

catch (**KeyNotFoundException**)

{ Console.WriteLine("La clef \"tif\" est introuvable."); }

// La méthode ContainsKey permet de savoir si la clef existe déjà ou non ==> true si clef existe

if **(!mesAppli.ContainsKey**("htm"))

{ mesAppli.Add("htm", "firefox.exe");

Console.WriteLine("Ajout de la clef \"htm\" avec la valeur {0}.",mesAppli["htm"]); }

// La méthode Remove permet de supprimer une paire clef/valeur.

Console.WriteLine("Essai de suppression de la clef \"doc\" et de la valeur associée.");

**mesAppli.Remove("doc");**

if (!mesAppli.ContainsKey("doc"))

{ Console.WriteLine("La clef \"doc\" est introuvable."); }

// Count indique le nombre de paires stockées.

Console.WriteLine("Ce dictionnaire contient {0} paires.", mesAppli.Count);

// Affichage des clés du dictionnaire

**Dictionary<string, string>.KeyCollection lesClésDeMaCollection = mesAppli.Keys;**

// On parcourt les clefs (qui sont des objets de type string).

Console.WriteLine("\nListe des clefs"); //Le \n sert à faire à sauter une ligne)

Console.WriteLine("-----------------");

foreach (string uneCle in lesClésDeMaCollection)

{ Console.WriteLine(uneCle); }

// Affichage des valeurs du dictionnaire

**Dictionary<string, string>.ValueCollection lesValeursDeMaCollection = mesAppli.Values;**

// On parcourt les clefs (qui sont des objets de type string).

Console.WriteLine("\nListe des valeurs");

Console.WriteLine("-----------------");

foreach (string uneValeur in lesValeursDeMaCollection)

{ Console.WriteLine(uneValeur); }

**Corrigé indicatif**

// Affichage du contenu du dictionnaire : clés et valeurs

// ces éléments sont récupérés en tant que des objets de type KeyValuePair.

//--------------------------------------------------------------------------

Console.WriteLine("\nAffichage du contenu du dictionnaire");

Console.WriteLine("-------------------------------------");

**foreach (KeyValuePair<string, string> pcv in mesAppli)**

{

Console.WriteLine("L'extension [\"{0}\"] est associée à {1}", pcv.Key, pcv.Value);

}

// On associe l'application winword.exe à la clé "docx"

**mesAppli["docx"] = "winword.exe";**

**Console.WriteLine("\nLa nouvelle valeur associée à la clef \"docx\" est {0}.", mesAppli["docx"]);**

// Affichage du contenu du dictionnaire : clés et valeurs

// ces éléments sont récupérés en tant que des objets de type KeyValuePair.

//--------------------------------------------------------------------------

Console.WriteLine("\nAffichage du contenu du dictionnaire");

Console.WriteLine("-------------------------------------");

**foreach (KeyValuePair<string, string> pcv in mesAppli)**

**{**

**Console.WriteLine("L'extension [\"{0}\"] est associée à {1}", pcv.Key, pcv.Value);**

**}**

// La méthode Clear permet de supprimer toutes les paires clef/valeur.

Console.WriteLine("\nAvant suppression, ce dictionnaire contient {0} paires.", mesAppli.Count);

**mesAppli.Clear();**

Console.WriteLine("\nAprès suppression, ce dictionnaire contient {0} paires.", mesAppli.Count);

}

}

}