Classes d’usage courantes

C#

String

`public int Length { get; }`	nombre de caractères de la chaîne
`public bool EndsWith(string value)`	rend vrai si la chaîne se termine par *value*
`public bool StartsWith(string value)`	rend vrai si la chaîne commence par *value*
`public virtual bool Equals(object obj)`	rend vrai si la chaînes est égale à *obj* - équivalent chaîne==obj
`public int IndexOf(string value, int startIndex)`	rend la première position dans la chaîne de la chaîne *value* - la recherche commence à partir du caractère n° *startIndex*
`public int IndexOf(char value, int startIndex)`	idem mais pour le caractère *value*
`public string Insert(int startIndex, string value)`	insère la chaîne *value* dans chaîne en position *startIndex*
`public static string Join(string separator, string[] value)`	méthode de classe - rend une chaîne de caractères, résultat de la concaténation des valeurs du tableau *value* avec le séparateur *separator*
`public int LastIndexOf(char value, int startIndex, int count)` `public int LastIndexOf(string value, int startIndex, int count)`	idem *indexOf* mais rend la dernière position au lieu de la première
`public string Replace(char oldChar, char newChar)`	rend une chaîne copie de la chaîne courante où le caractère *oldChar* a été remplacé par le caractère *newChar*
`public string[] Split(char[] separator)`	la chaîne est vue comme une suite de champs séparés par les caractères présents dans le tableau *separator*. Le résultat est le tableau de ces champs
`public string Substring(int startIndex, int length)`	sous-chaîne de la chaîne courante commençant à la position *startIndex* et ayant *length* caractères
`public string ToLower()`	rend la chaîne courante en minuscules
`public string ToUpper()`	rend la chaîne courante en majuscules
`public string Trim()`	rend la chaîne courante débarrassée de ses espaces de début et fin

string

- Une chaine C peut être considérée comme un tableau de caracteres:
  - C[i] est le caractere i de C
  - C.Length est le nombre de caracteres de C
- Exemples

tableaux

Propriétés

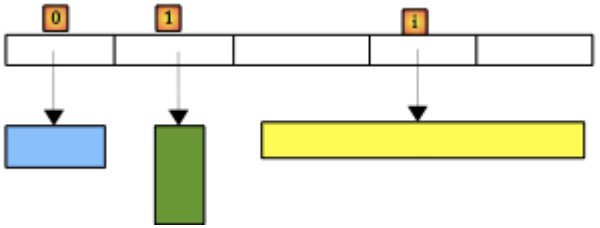
`public int Length {get;}`	nombre total d'éléments du tableau, quelque soit son nombre de dimensions
`public int Rank {get;}`	nombre total de dimensions du tableau

Méthodes

`public static int BinarySearch<T>(T[] tableau,T value)`	rend la position de value dans tableau.
`public static int BinarySearch<T>(T[] tableau,int index, int length, T value)`	idem mais cherche dans tableau à partir de la position index et sur length éléments
`public static void Clear(Array tableau, int index, int length)`	met les length éléments de tableau commençant au n° index à 0 si numériques, false si booléens, null si références
`public static void Copy(Array source, Array destination, int length)`	copie length éléments de source dans destination
`public int GetLength(int i)`	nombre d'éléments de la dimension n° i du tableau
`public int GetLowerBound(int i)`	indice du 1er élément de la dimension n° i
`public int GetUpperBound(int i)`	indice du dernier élément de la dimension n° i
`public static int IndexOf<T>(T[] tableau, T valeur)`	rend la position de valeur dans tableau ou -1 si valeur n'est pas trouvée.
`public static void Resize<T>(ref T[] tableau, int n)`	redimensionne tableau à n éléments. Les éléments déjà présents sont conservés.
`public static void Sort<T>(T[] tableau, IComparer<T> comparateur)`	trie tableau selon un ordre défini par comparateur. Cette méthode a été présentée page 81.

Listes génériques et mixtes

- Listes mixtes: ArrayList:
  - Les différents objets sont encapsules dans des références a des objets standards (boxing)



  - ArrayList L=new ArrayList();
    - L.add(4)
    - L'entier 4 est encapsule dans un objet générique

Listes génériques et mixtes

- Pour l'utilisation:
  - Int i=(int)L[0];
- Le cast est le unboxing.
- Si les objets sont identiques il vaut mieux utiliser des listes génériques

- Liste générique: un objet List<T> ou T est une classe

Listes génériques et mixtes

Propriétés

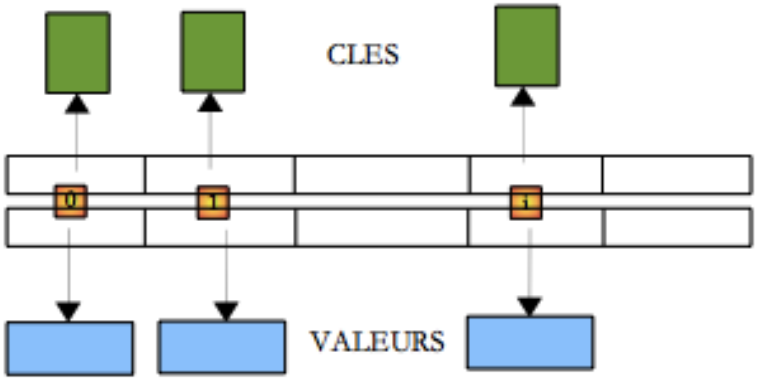
`public int Count {get;}`	nombre d'éléments de la liste
`public int Capacity {get;}`	nombre d'éléments que la liste peut contenir avant d'être redimensionnée. Ce redimensionnement se fait automatiquement. Cette notion de capacité de liste est analogue à celle de capacité décrite pour la classe StringBuilder page 95.

Méthodes

`public void Add(T item)`	ajoute item à la liste
`public int BinarySearch<T>(T item)`	rend la position de item dans la liste s'il s'y trouve sinon un nombre <0
`public int BinarySearch<T>(T item, IComparer<T> comparateur)`	idem mais le 2ième paramètre permet de comparer deux éléments de la liste. L'interface IComparer<T> a été présentée page 81.
`public void Clear()`	supprime tous les éléments de la liste
`public bool Contains(T item)`	rend True si item est dans la liste, False sinon
`public void CopyTo(T[] tableau)`	copie les éléments de la liste dans tableau.
`public int IndexOf(T item)`	rend la position de item dans tableau ou -1 si valeur n'est pas trouvée.
`public void Insert(T item, int index)`	insère item à la position index de la liste
`public bool Remove(T item)`	supprime item de la liste. Rend True si l'opération réussit, False sinon.
`public void RemoveAt(int index)`	supprime l'élément n° index de la liste
`public void Sort(IComparer<T> comparateur)`	trie la liste selon un ordre défini par comparateur. Cette méthode a été présentée page 81.
`public void Sort()`	trie la liste selon l'ordre défini par le type des éléments de la liste
`public T[] ToArray()`	rend les éléments de la liste sous forme de tableau

La classe dictionnary<Tkey,Tvalue>

- Un dictionnaire est un tableau a deux colonnes avec des clés et des valeurs. Une clé est unique.



La classe dictionnary<Tkey,Tvalue>

Constructeurs

`public Dictionary<TKey,TValue>()`	constructeur sans paramètres - construit un dictionnaire vide. Il existe plusieurs autres constructeurs.

Propriétés

`public int Count {get;}`	nombre d'entrées (clé, valeur) dans le dictionnaire
`public Dictionary<TKey,TValue>.KeyCollection Keys {get;}`	collection des clés du dictionnaire.
`public Dictionary<TKey,TValue>.ValueCollection Values {get;}`	collection des valeurs du dictionnaire.

Méthodes

`public void Add(TKey key, TValue value)`	ajoute le couple (key, value) au dictionnaire
`public void Clear()`	supprime tous les couples du dictionnaire
`public bool ContainsKey (TKey key)`	rend True si key est une clé du dictionnaire, False sinon
`public bool ContainsValue (TValue value)`	rend True si value est une valeur du dictionnaire, False sinon
`public void CopyTo(T[] tableau)`	copie les éléments de la liste dans tableau.
`public bool Remove(TKey key)`	supprime du dictionnaire le couple de clé key. Rend True si l'opération réussit, False sinon.
`public bool TryGetValue(TKey key,out TValue value)`	rend dans value, la valeur associée à la clé key si cette dernière existe, sinon rend la valeur par défaut du type TValue (0 pour les nombres, false pour les booléens, null pour les références d'objet)

Lecture des fichiers textes

Constructeurs

public StreamReader(string path)	construit un flux de lecture à partir du fichier de chemin *path*. Le contenu du fichier peut être encodé de diverses façons. Il existe un constructeur qui permet de préciser le codage utilisé. Par défaut, c'est le codage UTF-8 qui est utilisé.

Propriétés

public bool EndOfStream {get;}	True si le flux a été lu entièrement

Méthodes

public void Close()	ferme le flux et libère les ressources allouées pour sa gestion. A faire obligatoirement après exploitation du flux.
public override int Peek()	rend le caractère suivant du flux sans le consommer. Un Peek supplémentaire rendrait donc le même caractère.
public override int Read()	rend le caractère suivant du flux et avance d'un caractère dans le flux.
public override int Read(char[] buffer, int index, int count)	lit *count* caractères dans le flux et les met dans *buffer* à partir de la position *index*. Rend le nombre de caractères lus - peut être 0.
public override string ReadLine()	rend la ligne suivante du flux ou null si on était à la fin du flux.
public override string ReadToEnd()	rend la fin du flux ou "" si on était à la fin du flux.

Ecrire dans des fichiers textes

Constructeurs

public StreamWriter(string path)	construit un flux d'écriture dans le fichier de chemin *path*. Le contenu du fichier peut être encodé de diverses façons. Il existe un constructeur qui permet de préciser le codage utilisé. Par défaut, c'est le codage UTF-8 qui est utilisé.

Propriétés

public virtual bool AutoFlush {get;set;}	fixe le mode d'écriture dans le fichier du buffer associé au flux. Si égal à **False**, l'écriture dans le flux n'est pas immédiate : il y a d'abord écriture dans une mémoire tampon puis dans le fichier lorsque la mémoire tampon est pleine sinon l'écriture dans le fichier est immédiate (pas de tampon intermédiaire). Par défaut c'est le mode tamponné qui est utilisé. Le tampon n'est écrit dans le fichier que lorsqu'il est plein ou bien lorsqu'on le vide explicitement par une opération **Flush** ou encore lorsqu'on ferme le flux **StreamWriter** par une opération **Close**. Le mode **AutoFlush=False** est le plus efficace lorsqu'on travaille avec des fichiers parce qu'il limite les accès disque. C'est le mode par défaut pour ce type de flux. Le mode **AutoFlush=False** ne convient pas à tous les flux, notamment les flux réseau. Pour ceux-ci, qui souvent prennent place dans un dialogue entre deux partenaires, ce qui est écrit par l'un des partenaires doit être immédiatement lu par l'autre. Le flux d'écriture doit alors être en mode **AutoFlush=True**.
public virtual string NewLine {get;set;}	les caractères de fin de ligne. Par défaut "\r\n". Pour un système Unix, il faudrait utiliser "\n".

Méthodes

public void Close()	ferme le flux et libère les ressources allouées pour sa gestion. A faire obligatoirement après exploitation du flux.
public override void Flush()	écrit dans le fichier, le buffer du flux, sans attendre qu'il soit plein.
public virtual void Write(T value)	écrit value dans le fichier associé au flux. Ici T n'est pas un type générique mais symbolise le fait que la méthode *Write* accepte différents types de paramètres (string, int, object, ...). La méthode *value.ToString* est utilisée pour produire la chaîne écrite dans le fichier.
public virtual void WriteLine(T value)	même chose que *Write* mais avec la marque de fin de ligne (NewLine) en plus.