Collections (suite)

- Trier un Array
- Tri personnalisé
- Trier une liste
- CompareTo
- Expression Lambda
- Filtrage d'une collection

Tri d'un `Array`

- Pour trier un tableau en C#, on utilise la méthode statique `Array.Sort()`.
- Le tri directement sur la collection, c'est-à-dire que le tableau d'origine est modifié.
- L'ordre de tri est croissant.

```
int[] nombres = [5, 2, 9, 1, 3];
Array.Sort(nombres);
Console.WriteLine(string.Join(", ", nombres)); // Résultat : 1, 2, 3, 5, 9
```

- Notez au passage la méthode string.Join() qui permet de concaténer toutes les valeurs d'un Array avec un séparateur.

Tri décroissant

- La méthode `Array.Reverse()` inverse les valeurs d'un array.
- Pour effectuer un tri décroissant, vous pouvez utiliser `Array.Reverse()` après avoir trié avec `Array.Sort()`.

```
int[] nombres = [5, 2, 9, 1, 3];
Array.Sort(nombres);
Array.Reverse(nombres);
Console.WriteLine(string.Join(", ", nombres)); // Résultat : 9, 5, 3, 2, 1
```

Tri d'une `List<T>`

- Pour trier une liste, le fonctionnement est sensiblement le même pour une liste que pour un Array, excepté qu'on utilise l'instance de la liste pour appeler les méthodes `Sort()` ou `Reverse()`

```
List<int> nombres = [5, 2, 9, 1, 3];
nombres.Reverse();
nombres.Sort();
Console.WriteLine(string.Join(", ", nombres)); // Résultat : 9, 5, 3, 2, 1
```

Tri personnalisé

- Pour personnaliser le tri, vous pouvez fournir une **expression lambda** à `Array.Sort()` qui définit la manière dont deux éléments doivent être comparés.
- Par exemple, nous pouvons trier les nombres en fonction de leur **valeur absolue**.

```
int[] nombres = [5, -2, 9, -1, 3];
Array.Sort(nombres, (a, b) => Math.Abs(a).CompareTo(Math.Abs(b)));
Console.WriteLine(string.Join(", ", nombres)); // Résultat : -1, -2, 3, 5, 9
```

Prenons le temps de décortiquer la fonction CompareTo et l'expression Lambda.

Qu'est-ce qu'un la méthode `CompareTo()`

- La méthode `CompareTo()` compare deux éléments et retourne :
  - Un nombre positif (1) si l'élément actuel est supérieur à l'autre.
  - 0 s'ils sont égaux.
  - Un nombre négatif (-1) si l'élément actuel est inférieur.

```
(5).CompareTo(9) // retourne -1
(5).CompareTo(5) // retourne 0
(9).CompareTo(5) // retourne 1
```

- La fonction CompareTo() est utilisée par les algorithmes de tri pour déterminer l'ordre des éléments.
- L'expression par défaut est `a.CompareTo(b)`, cela va trier en ordre croissant.
- Si on utilise l'expression: b.Compare(a), cela va trier en ordre décroissant.

Qu'est-ce qu'une expression lambda ?

- Une **expression lambda** est une manière concise d'écrire une fonction anonyme (une fonction sans nom) qui peut être passée comme paramètre à une méthode.
- Cela permet de spécifier une logique très simple en une seule ligne.
    - La portion à gauche de la flèche `=>` représente le ou les paramètres..
    - La portion à droite représente ce qui sera retourné par la fonction anonyme.

Exemple : `n => Math.Abs(n)`

- Ici, `n => Math.Abs(n)` est une **expression lambda**.
- Elle signifie : "**pour chaque élément `n`, retourne la valeur absolue de `n`**".
- C'est l'équivalent d'une fonction qui retourne la valeur absolue.

```
int anonyme(int n){
    return Math.Abs(n);
}
```

Utilisation dans `Array.Sort()` et `List.Sort()`

- Ainsi, pour revenir à notre exemple précédent, nous pouvons ajouter à la méthode Sort() une **expression lambda** qui compare deux éléments afin de personnalisé notre méthode de tri.
- Par exemple, dans `(a, b) => Math.Abs(a).CompareTo(Math.Abs(b))`, on compare la valeur absolue de deux nombres `a` et `b`. Le tri s'effectue ensuite selon ces valeurs absolues.

```
int[] nombres = [5, -2, 9, -1, 3];
Array.Sort(nombres, (a, b) => Math.Abs(a).CompareTo(Math.Abs(b)));
Console.WriteLine(string.Join(", ", nombres)); // Résultat : -1, -2, 3, 5, 9
```

Filtrage avec `Array`

- Pour filtrer un tableau en fonction d'une condition, on peut utiliser `Array.FindAll()`.
- Cette méthode renvoie un nouveau tableau contenant uniquement les éléments qui respectent une condition.
- Nous devons passer en paramètre à la méthode FindAll l'expression Lambda qui indique quoi filtrer.
    - L'expression doit retourner vrai ou faux et sera vérifié pour chaque valeur de la collection afin de déterminer ce qui fera partie du nouveau tableau filtré.

Exemple : Trouver tous les nombres pairs

```
int[] nombres = [5, 2, 9, 1, 3];
int[] nombresPairs = Array.FindAll(nombres, n => n % 2 == 0);
Console.WriteLine(string.Join(", ", nombresPairs)); // Résultat : 2
```

Exemple : Trouver tous nom avec un 'i'

```
string[] noms = ["bob", "Alice", "Marcel"];
string[] nomAvecI = Array.FindAll(noms, n => n.contains("i"));
Console.WriteLine(string.Join(", ", nombresPairs)); // Résultat : Alice
```

Utilisation dans `Array.FindAll()`

- La méthode `Array.FindAll()` prend deux paramètres :
    i. Le tableau à filtrer (`nombres`).
    ii. Une expression lambda qui dit comment filtrer (`n => n % 2 == 0`).
- Pour chaque élément du tableau, la fonction est appelée, et seuls les éléments qui remplissent la condition sont ajoutés dans le nouveau tableau.

Filtrage avec `List<T>`

- Pour les listes, la méthode `FindAll()` fonctionne de manière similaire à celle des tableaux.

Exemple : Filtrer une liste de nombres pairs

```
List<int> nombres = [5, 2, 9, 1, 3];
List<int> nombresPairs = nombres.FindAll(n => n % 2 == 0);
Console.WriteLine(string.Join(", ", nombresPairs)); // Résultat : 2
```

Résumé

Tri

- **Tri d'un tableau (`Array`)** : Utilisez `Array.Sort(monArray)` pour trier en place. Pour un tri personnalisé, passez une expression lambda qui définit comment comparer deux éléments.
- **Tri d'une liste (`List<T>`)** : Utilisez `maListe.Sort()` pour trier en place. Vous pouvez aussi personnaliser le tri avec une expression lambda.

Expressions lambda

- Elles permettent de passer des conditions ou des comparaisons simples sous forme de fonctions anonymes, par exemple : `(a, b) => b.CompareTo(a)` pour trier en ordre décroissant.

Résumé (suite)

Méthode `CompareTo()`

- Utilisée pour comparer deux valeurs. Elle retourne un nombre positif, 0 ou un nombre négatif selon si l'élément actuel est plus grand, égal ou plus petit que l'autre.

Filtrage

- **Filtrage d'un tableau ou d'une liste** : Utilisez `Array.FindAll()` ou `List.FindAll()` en passant une **expression lambda** pour spécifier la condition de filtrage.