Support de cours LINQ TO OBJECTS

Comment était la vie avant LINQ ?

Hétérogénéité de la manipulation de données

LINQ = Langage Integrated Query

Proposer des methodes pour manipuler les données de manière homogène où qu'elles soient stockées

Les requetes LINQ peuvent s'executer sur des Collections, XML, Base de données


Objects
MongoDB
CSV Files
File System
LINQ is Everywhere
SQL Database
HL7 XML
JSON

Démo avec et sans LINQ

Programme

1. Fondations
2. Requetes basiques
3. Filtrage, Tri, Projection
4. Groupby, Join, Aggregation
5. Application sur un fichier XML

LINQ
Fondations

```
IEnumerable<string> filteredList =
                cities.Where(StartsWithL);

public bool StartsWithL(string name)
{
    return name.StartsWith("L");
}
```

Named Method

```
IEnumerable<string> filteredList =
    cities.Where(delegate(string s)
                        { return s.StartsWith("L"); });
```

Anonymous Method

```
IEnumerable<string> filteredList =
    cities.Where(s => s.StartsWith("L"));
```

Lambda expressions

```
results.Where(f=>f.Length > 100)
```

- Une *expression lambda* est un bloc de code (une expression ou un bloc d'instructions) qui est traité comme un objet.

- Elle peut être passée comme argument à des méthodes, et peut aussi être retournée par des appels de méthode.

- Une expression lambda comportant une expression à droite de l'opérateur **=>** est appelée expression lambda

Pattern et exemples

- (inputparameters) => expression

```
results.Where(f=>f.Length > 100)
```

X=>X+1

(X,Y) => X*Y

(int x, string s) => s.Length > x;

X => Console.WriteLine(X);

Quel est le type d'une expression ?

- Func<*input*,*output*>
- Func<*input1*, *input2*, *output*> (jusqu'à 16 inputs mais 1 output)

- Action ne revoit pas de résultat

- Predicate = Func<*input*,bool>

```
Func<int,int> expression = x => x + 1;
list.Select(expression );

Action<int> action =
x=>Console.WriteLine(x);
```

Requetes LINQ

Quel objet peut utiliser LINQ ?

- Les méthodes de LINQ sont accessibles à tout objet qui implémente **Ienumerable<T>**

- Ex : List, Array,Queue,Stack …

Méthodes disponibles

- **Where**: permet de filtrer les résultats

```
var filteredResults = results.Where(f => f.Length > 100);
```

- **FirstOrDefault** : Permet de récupérer 1 objet.

```
var filteredResult = results.FirstOrDefault(f => f.Name =="toto");
```

- **Select** : Permet de faire des projections et transformations

```
var list2 = list.Select(x => x + 1);
```

Exercice 1

- *var list = Enumerable.Range(0, 500).ToList();*

1. Ne récupérer que les valeurs paires
2. Récupérer que la valeur 250
3. Diviser par 2 toutes les valeurs

Autres méthodes

- Average() : calcule une moyenne
- Count() : calcule le nombre d'elements
- Take, TakeWhile() : prend les elemetns tant que la condition est vraie
- Sum() : fait une somme
- OrderBy(), OrderByDescending() : tri
- Min(),Max() : obtient la valeur min ou max

Exercice 2

var list = Enumerable.Range(0, 500).ToList();

1. Quel est le nombre d'éléments qui sont des multiples de 3 ?
2. Quelle est la somme des élements supérieurs à 20 ?
3. Quelles est la moyenne des multiples de 3 ?
4. Inverser l'ordre des données de 2 manières différentes
5. Quelle est la valeur minimale de la liste ?

Aggregate : Autre méthode LINQ

- Aggregate() : Fait une opépration sur chaque élement de la liste en prenant en compte les opérations effectuées

```
var nums = new[]{1,2,3,4};
var sum = nums.Aggregate( (a,b) => a + b);
Console.WriteLine(sum); // output: 10 (1+2+3+4)
```

```
var chars = new []{"a","b","c", "d"};
var csv = chars.Aggregate( (a,b) => a + ',' + b);
Console.WriteLine(csv); // Output a,b,c,d
```

```
var multipliers = new []{10,20,30,40};
var multiplied = multipliers.Aggregate(5, (a,b) => a * b);
Console.WriteLine(multiplied); //Output 1200000 ((((5*10)*20)*30)*40)
```

Zip

- Zip() : Fusionne 2 séquences d'élements avec une fonction de selection

```
var letters= new string[] { "A", "B", "C", "D", "E" };
var numbers= new int[] { 1, 2, 3 };
var q = letters.Zip(numbers, (l, n) => l + n.ToString());
foreach (var s in q)
    Console.WriteLine(s);
```

Ouput

```
A1
B2
C3
```

Exercice Zip ou Aggregate

```
var array1 = new int[] { 1, 2, 3, 4, 5 };
var array2 = new int[] { 6, 7, 8, 9, 10 };
```

1) Générer un tableau {7,9,11,13,15} qui fait la somme une à une