Accumulateur | Aggrégateur | Reduce

Cet aspect a déjà été aperçu avec les fonctions `Sum` et `Average` utilisées pour l'exercice Epsilon.

Voici un exemple basique pour calculer une somme et une moyenne:

```
List<int> numbers = new(){1,2,3,4,5};
int sum = numbers.Sum(); // 15
double average = numbers.Average(); // 3.0
```

> Ces 2 fonctions effectuent une opération qui "aplatit" la liste en la **réduisant** (d'où le terme `Reduce`) à une seule valeur.

À cela s'ajoutent deux autres fonctions bien pratiques, `Min` et `Max`:

```
int max = numbers.Max(); // 5
int min = numbers.Min(); // 1
```

Où est la programmation fonctionnelle ici ?

Jusque là, on a l'impression d'avoir à faire à des fonctions standards, on va donc étudier un nouveau cas pour lequel on comprend bien l'intérêt d'avoir des fonctions de réduction d'ordre supérieur:

```
class Person{
    public string Name{get;set;}
    public int Age{get;set;}
    public int Sisters{get;set;}
    public int Brothers{get;set;}
  }

List<Person> cid5d = new List<Person>(){
    new Person(){Name="Paul",Age=15,Sisters=2,Brothers=1},
    new Person(){Name="Lucie",Age=18,Sisters=1,Brothers=3},
    new Person(){Name="Claude",Age=16,Sisters=0,Brothers=0}
};

double averageAge = cid5d.Average(person=>person.Age);
double averageSiblings = cid5d.Average(person=>person.Sisters + person.Brothers);

int minAge = cid5d.Min(person=>person.Age);
```

Aggrégateur générique

Outre les accumulateurs particuliers fournis par *LINQ*, il existe une fonction d'ordre supérieur générique pour la réduction nommée `Aggregate` dont voici un premier exemple:

```
int sum = numbers.Aggregate((current,next)=>current+next)
```

Chaque élément est comparé à celui d'après et en résulte un seul élément défini par le lambda. Ainsi, à la fin de l'opération, *il ne doit en rester qu'un*...

Réécriture de `Min`

Avec l'accumulateur générique `Aggregate`, on peut réécrire le `Min` ainsi:

```
int min = numbers.Aggregate((a,b)=>Convert.ToInt32(Math.Min(a,b))); //1
```

Et ainsi de suite pour les autres opérateurs.

Aggrégateurs avec classes

Pour des types non primitifs, on doit utiliser une signature plus complète avec 3 éléments:

- Seed (valeur de départ à comparer avec le 1er élément)
- Fonction d'aggrégation
- Choix de la forme du résultat

```
var min = cid5d.Aggregate(
    new Person(){Brothers=99}, //Seed
    (a,b)=>a.Brothers<b.Brothers?a:b, //Min logic
    person=>person.Name); //Result transformer
```

Que vaut min ?

- 0
- Paul
- Claude
- 1
- ...

La fonction d'aggrégation sélectionne la personne avec le moins de frères et la forme du résultat est demandée sous forme du nom de la personne.

Le résultat est donc:

▶ Cliquer ici pour voir/vérifier la réponse