Zadanie (LINQ-to-Objects) - Sortowanie wg nazwisk, później wg imion

- Krzysztof Molenda, v.01

Problem

Dany jest napis składający się z imion i nazwisk oddzielonych przecinkami, np.
`"Krzysztof Molenda, Jan Kowalski, Anna Abacka , Józef Kabacki, Kazimierz Moksa"`

Napisz zapytanie, które zwróci ten napis posortowany według nazwisk, a następnie według imion.

> Zaprojektuj zapytanie LINQ, staraj się myśleć deklaratywnie.
>
> Najpierw skoncentruj się na jakimkolwiek rozwiązaniu problemu, później optymalizuj zapytanie tak, aby było jak najkrótsze i jak najszybsze, najlepiej bez materializowania wyników cząstkowych.

Rozwiązanie *Krok-Po-Kroku*

Poniżej podano rozwiązanie krok-po-kroku, od wariantu naiwnego do ostatecznego - z wykorzystaniem *Ling-to-object*.

Krok 1.1 - rozwiązanie naiwne (bez LINQ)

```
string s = "Krzysztof Molenda,  Jan Kowalski, Anna Abacka , Józef Kabacki, Kazimierz Moksa";
```

Podpowiedź: do rozbicia napisu na elementy oddzielone separatorem (tokeny) służy metoda `Split` klasy `string`.

```
string[] osoby = s.Split(',');
foreach( var osoba in osoby)
    Console.WriteLine( osoba );
```

```
Krzysztof Molenda
  Jan Kowalski
 Anna Abacka
 Józef Kabacki
 Kazimierz Moksa
```

Pierwszy dostrzeżony problem - należy usunąć spacje przed i po tokenie. Zakładamy bowiem, że imiona i nazwiska oddzielone są przecinkiem (nie wiemy, iloma spacjami).

```
string[] osoby = s.Split(',');
for(int i = 0; i < osoby.Length; i++)
    osoby[i] = osoby[i].Trim();

foreach( var osoba in osoby)
    Console.WriteLine( osoba );
```

```
Krzysztof Molenda
Jan Kowalski
Anna Abacka
Józef Kabacki
Kazimierz Moksa
```

Teraz wypadałoby rozbić elementy tablicy na oddzielnie imiona i nazwiska. Problem - gdzie zapamiętać?

Tworzymy roboczą klasę `Osoba` zawierającą dwie właściwości: `Imie` oraz `Nazwisko` - możliwie jak najprostszą.

```
public class Osoba
{
    public string Imie {get; set; }
    public string Nazwisko {get; set; }
    public override string ToString() => $"({Imie}; {Nazwisko})";
}
```

Teraz wyniki parsowania zapamiętamy w liście osób:

```
var listaOsob = new List<Osoba>();

string[] osoby = s.Split(',');
for(int i = 0; i < osoby.Length; i++)
{
    osoby[i] = osoby[i].Trim();
    string[] temp = osoby[i].Split(' ');
    Osoba o = new Osoba();
    o.Imie = temp[0];
    o.Nazwisko = temp[1];
    listaOsob.Add( o );
}
```

Sprawdźmy, co przechowuje lista:

```
foreach( var osoba in listaOsob)
    Console.WriteLine( osoba );
```

```
(Krzysztof; Molenda)
(Jan; Kowalski)
(Anna; Abacka)
(Józef; Kabacki)
(Kazimierz; Moksa)
```

Teraz wystarczy ją przesortować i zwrócić wynik odpowiednio formatując:

```
listaOsob.Sort( (o1, o2) => String.Compare( o1.Nazwisko+o1.Imie, o2.Nazwisko+o2.Imie ) );

string wynik = "";
foreach(var x in listaOsob)
    wynik += x.Imie + " " + x.Nazwisko + ", ";

Console.WriteLine( wynik );
```

```
Anna Abacka, Józef Kabacki, Jan Kowalski, Kazimierz Moksa, Krzysztof Molenda,
```

□

Krok 1.2 - rozwiązanie naiwne (bez LINQ) - `Tuple`

Skracamy kod. Wyeliminujmy klasę `Osoba`. Zastąpić ją możemy klasą `Tuple`.

```
var listaOsob = new List< Tuple<string, string> >();

string[] osoby = s.Split(',');
for(int i = 0; i < osoby.Length; i++)
{
    osoby[i] = osoby[i].Trim();
    string[] temp = osoby[i].Split(' ');
    var o = new Tuple<string, string>(temp[0], temp[1]);
    listaOsob.Add( o );
}

listaOsob.Sort( (o1, o2) => String.Compare( o1.Item2+o1.Item1, o2.Item2+o2.Item1 ) );

string wynik = String.Join(", ", listaOsob);

Console.WriteLine( wynik );
```

```
(Anna, Abacka), (Józef, Kabacki), (Jan, Kowalski), (Kazimierz, Moksa), (Krzysztof, Molenda)
```

Wydruk uwzględnia domyślne formatowanie obiektów typu `Tuple` (zdefiniowane w `ToString()` tej klasy) - elementy wypisywane są w nawiasach i oddzielone przecinkami.

□

Krok 1.2 - rozwiązanie naiwne (bez LINQ) - typy i obiekty anonimowe

W C# 7.x możemy użyć obiektów anonimowych. W tym przykładzie będzie to trochę sztuczne i trzeba zastosować *trick* polegający na stworzeniu listy na bazie przykładu (technika *casting by example*). (Ponieważ klasy anonimowe pochodzą bezpośrednio od typu `Object` i nie można ich rzutować na inne typy, nie możemy więc zbudować listy obiektów anonimowych).

```
//trick - begin
var OsobaExample = new {imie="", nazwisko=""};
var listaOsob = (new[] {OsobaExample}).ToList(); //trick
listaOsob.RemoveAt(0); //lista jest pusta
//trick - end

string[] osoby = s.Split(',');
```

```
for(int i = 0; i < osoby.Length; i++)
{
    osoby[i] = osoby[i].Trim();
    string[] temp = osoby[i].Split(' ');
    var o = new { imie = temp[0], nazwisko = temp[1] };
    listaOsob.Add( o );
}

listaOsob.Sort( (o1, o2) => String.Compare( o1.nazwisko+o1.imie, o2.nazwisko+o2.imie ) );

string wynik = String.Join(", ", listaOsob);

Console.WriteLine( wynik );
```

```
{ imie = Anna, nazwisko = Abacka }, { imie = Józef, nazwisko = Kabacki }, { imie = Jan, nazwisko = Kowalski }, { imie = Kazimierz, nazwisko =
```

Tym razem wydruk uwzględnia formatowanie obiektów anonimowych.

□

Krok 2.1 - rozwiązanie z LINQ

Wprowadzamy LINQ:

```
using System.Linq;

// Main()
Console.OutputEncoding = System.Text.Encoding.UTF8;
string s = "Krzysztof Molenda,  Jan Kowalski, Anna Abacka , Józef Kabacki, Kazimierz Moksa";

var query1 = s.Split(','); //rozbijamy na osoby
var query2 = query1.Select( osoba => osoba.Trim() ); //usuwamy spacje

// drukujemy
    query2.ToList().ForEach( x => { Console.Write( x + ", " ); } );
    Console.WriteLine();

var query3 = query2
                .Select( osoba => osoba.Split(' ') )
                .Select( x => (imie: x[0], nazwisko: x[1]) );
// drukujemy
    query3.ToList().ForEach( x => { Console.Write( x + ", " ); } );
    Console.WriteLine();

var query4 = query3
                .OrderBy( o => o.nazwisko )
                .ThenBy( o => o.imie );
// drukujemy
    query4.ToList().ForEach( x => { Console.Write( x + ", " ); } );
    Console.WriteLine();

var query5 = query4.Select( o => o.imie + " " + o.nazwisko );
//drukujemy
    query5.ToList().ForEach( x => { Console.Write( x + ", " ); } );
    Console.WriteLine();

// ostatecznie
string wynik = String.Join(", ", query5);
Console.WriteLine( wynik );
```

```
Krzysztof Molenda, Jan Kowalski, Anna Abacka, Józef Kabacki, Kazimierz Moksa,
(Krzysztof, Molenda), (Jan, Kowalski), (Anna, Abacka), (Józef, Kabacki), (Kazimierz, Moksa),
(Anna, Abacka), (Józef, Kabacki), (Jan, Kowalski), (Kazimierz, Moksa), (Krzysztof, Molenda),
Anna Abacka, Józef Kabacki, Jan Kowalski, Kazimierz Moksa, Krzysztof Molenda,
Anna Abacka, Józef Kabacki, Jan Kowalski, Kazimierz Moksa, Krzysztof Molenda
```

□

Krok 2.2 - rozwiązanie z LINQ - jedna kwerenda, notacja *flow*

```
using System.Linq;

//...

Console.OutputEncoding = System.Text.Encoding.UTF8;
string s = "Krzysztof Molenda,  Jan Kowalski, Anna Abacka , Józef Kabacki, Kazimierz Moksa";

var query = s.Split(',')                                    //rozbijamy na osoby -> tablica napisów
             .Select( osoba => osoba.Trim() )               //usuwamy spacje
             .Select( osoba => osoba.Split(' ') )           //rozbijamy na wyrazy -> sekwencja tablic 2-elemento
             .Select( x => (imie: x[0], nazwisko: x[1]) )   //sekwencja obiektów anonimowych (imie, nazwisko)
             .OrderBy( o => o.nazwisko )                    //sortujemy wg nazwiska,
             .ThenBy( o => o.imie )                         //... a następnie wg imienia
             .Select( o => o.imie + " " + o.nazwisko );     //sekwencja naspisów postaci imie-spacja-nazwisko
```

```
string wynik = String.Join(", ", query);                    //łączymy sekwencję z przecinkami
Console.WriteLine( wynik );
```

```
Anna Abacka, Józef Kabacki, Jan Kowalski, Kazimierz Moksa, Krzysztof Molenda
```

□

Krok 2.3 - rozwiązanie z LINQ - notacja *query*

```
using System.Linq;

// ...

Console.OutputEncoding = System.Text.Encoding.UTF8;
string s = "Krzysztof Molenda,  Jan Kowalski, Anna Abacka , Józef Kabacki, Kazimierz Moksa";

var query = s.Split(',')
             .Select( x => x.Trim() )
             .Select( x => x.Split(' ') );
             //.Select( x => (imie: x[0], nazwisko: x[1]) ) ;

var query1 =
    from osoba in query
    let imie = osoba[0]
    let nazwisko = osoba[1]
    orderby nazwisko, imie
    select new {imie, nazwisko};

//string wynik = "";
//foreach( var osoba in query1 )
//    wynik += osoba.imie + " " + osoba.nazwisko + ", ";

string wynik = String.Join(", ", query1.Select( o => o.imie + " " + o.nazwisko ) );
Console.WriteLine( wynik );
```

```
Anna Abacka, Józef Kabacki, Jan Kowalski, Kazimierz Moksa, Krzysztof Molenda
```

□

Zadania

Wzorując się na powyższym przykładzie rozwiąż zadania w dwóch wariantach: w notacji *flow* oraz w notacji *query*.

Zadanie 1 -- Lista osób wg wieku

Dany jest napis składający się z imion, nazwisk i dat urodzenia, oddzielonych średnikami, np.

```
"Krzysztof Molenda, 1965-11-20; Jan Kowalski, 1987-01-01; Anna Abacka, 1972-05-20; Józef Kabacki, 2000-01-02; Kazimierz Moksa, 2001-
```

Napisz uniwersalne zapytanie zwracające nazwisko, imię oraz datę urodzenia w kolejności wg wieku, a następnie wg nazwiska.

Zadanie 2 -- Inicjały

Dany jest napis składający się z imion i nazwisk oddzielonych przecinkami, np.

```
"Krzysztof Molenda, Jan Kowalski,  Anna    Abacka, Józef Kabacki, Kazimierz Moksa,  Alfred Alacki"
```

Zapisz uniwersalny kod, który dla takiego napisu wypisze wszystkie grupy osób o tych samych inicjałach (liczebność grupy > 1).

Zadanie wykonaj bez Linq oraz z Linq (w notacji *flow* i w notacji *query*).