

“Programación .Net – Listas y Colecciones”

Actividades:

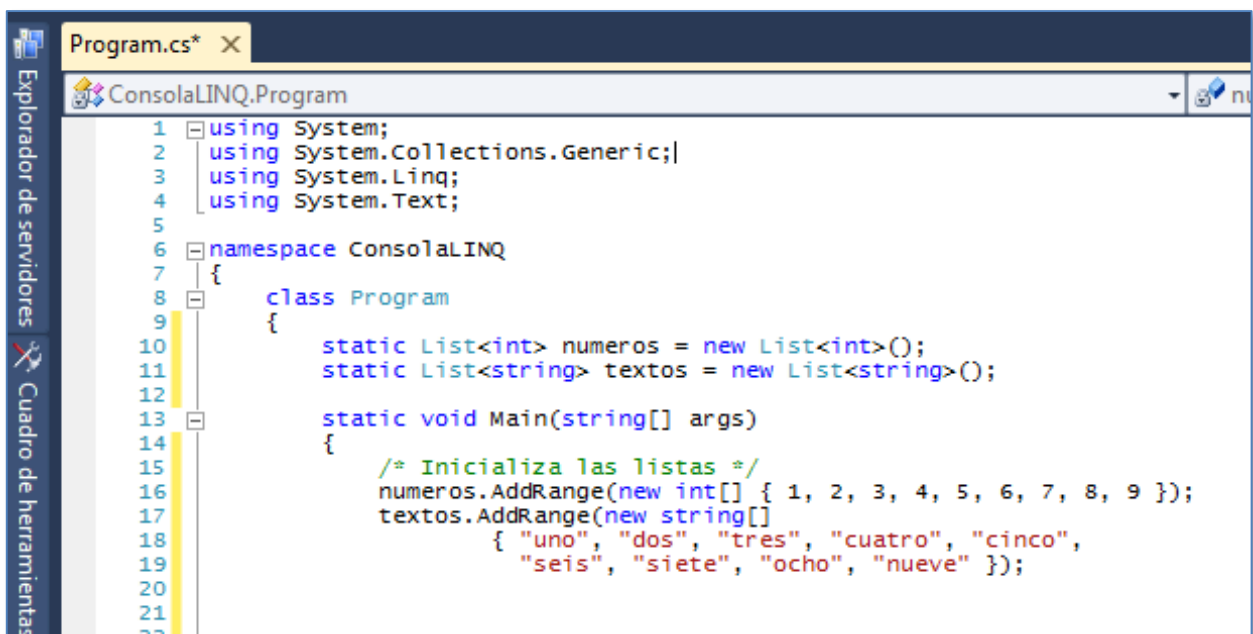
Consultas con LINQ.....	1
Elementos iniciales:	1
Menú de Opciones.....	1
Recuperación de valores con Select	2
Recuperación de valores con el filtro Where	3
Funciones agregadas Count y Sum	4
Funciones agregadas Min y Max.....	5
Función agregada para el promedio Average.....	6

Consultas con LINQ

Crearemos una aplicación de consola que nos permita ejercitar los distintos tipos de consulta que podemos realizar utilizando la sintaxis LINQ, para consultar colecciones de datos.

Elementos iniciales:

- Cree un solución de nombre “**Solucion Consultas LINQ**”
- Agregamos una nueva aplicación de consola de nombre “**ConsolaLINQ**”.
- Declaremos 2 listas genéricas como variables estáticas de la clase con los siguientes valore:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace ConsolaLINQ
7 {
8     class Program
9     {
10         static List<int> numeros = new List<int>();
11         static List<string> textos = new List<string>();
12
13         static void Main(string[] args)
14         {
15             /* Inicializa las listas */
16             numeros.AddRange(new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 });
17             textos.AddRange(new string[]
18                 { "uno", "dos", "tres", "cuatro", "cinco",
19                 "seis", "siete", "ocho", "nueve" });
20         }
21     }
22 }
```

Menú de Opciones

En el proyecto de consola creamos un menú con las siguientes opciones de trabajo:

1. Ejemplo Select
2. Ejemplo Where
3. Ejemplo Count y Sum
4. Ejemplo Min y Max
5. Ejemplo Average
6. Salir

Recuperación de valores con Select

- a) Para la opción 1 agregamos un método de nombre **EjemploSelect()**.
- b) Primero se desplegará el resultado de una consulta de selección simple de ambas listas con la cláusula **Select**.

```
Console.WriteLine("Consultas Select");

/* Numeros */
Console.WriteLine("Numeros");
var consulta1 = from n in numeros select n;
foreach (int item in consulta1)
{
    Console.Write("{0} ", item);
}
Console.WriteLine();

/* Textos */
Console.WriteLine("Textos");
var consulta2 = from txt in textos select txt;
foreach (string item in consulta2)
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

- c) Luego se desplegará el resultado de una consulta de selección con transformación, los números serán múltiplos de 10 y los textos se convertirán a mayúsculas, pero esta última se ejecuta con sintaxis de método:

```
Console.WriteLine("Select con Transformación");

/* Numeros */
Console.WriteLine("Numeros Múltiplos");
var consulta3 = from n in numeros select n * 10;
foreach (int item in consulta3)
{
    Console.Write("{0} ", item);
}
Console.WriteLine();

/* Textos */
Console.WriteLine("Textos en Mayúscula");
foreach (string item in textos.Select(t => t.ToUpper()))
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

Recuperación de valores con el filtro Where

- a) Para la opción 2 agregamos un método de nombre **EjemploWhere()**.
- b) Primero consultaremos los valores pares de la lista de números con la cláusula **Where**.

```
Console.Clear();
Console.WriteLine("Consultas Where");

/* Numeros */
Console.WriteLine("Numeros Pares");
var consulta1 = from n in numeros where n%2 == 0 select n;
foreach (int item in consulta1)
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

- c) Para los textos obtendremos solo aquellos de largo mayor a 5 caracteres en sintaxis de método.

```
/* Textos */
Console.WriteLine("Textos Mayores a 5 caracteres ");
foreach (string item in textos.Where(t => t.Length >= 5))
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

- d) Finalmente combinaremos el método **Where** y **Select**, para obtener el resultado anterior en mayúscula.

```
/* Textos */
Console.WriteLine("Textos Mayores a 5 caracteres en mayuscula");
foreach (string item in textos.Where(t => t.Length >= 5).Select(t => t.ToUpper()))
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

Funciones agregadas Count y Sum

- a) Para la opción 3 agregamos un método de nombre **EjemploCountSum()**.
- b) Primero obtendremos la cantidad de números impares y el resultado de su suma con los métodos **Count** y **Sum** en el resultado de una consulta.

```
Console.Clear();
Console.WriteLine("Funciones Count y Sum");

/* Numeros Impares */
var consulta1 = from n in numeros where n%2 != 0 select n;
Console.WriteLine("Numeros Impares : {0}", consulta1.Count());
Console.WriteLine("Suma de Impares : {0}", consulta1.Sum());
```

- c) Luego podemos contar cuantos textos tiene exactamente 5 caracteres, directo por medio de una sintaxis de método.

```
/* Textos de 5 caracteres */
Console.WriteLine("Textos de 5 caracteres : {0}", textos.Count(t => t.Length == 5));
```

- d) Finalmente podemos obtener la cantidad de textos con el carácter 'e' y luego haremos la combinación con el método **Select** para poder sumar el total de caracteres de estos textos.

```
/* Textos con la letra 'e' */
Console.WriteLine("Textos con 'e' : {0}", textos.Where(t => t.Contains('e')).Count());
Console.WriteLine("Suma de caracteres con 'e' : {0}",
    textos.Where(t => t.Contains('e')).Select(t => t.Length).Sum());
```

Funciones agregadas Min y Max

- a) Para la opción 4 agregamos un método de nombre **EjemploMinMax()**.
- b) Obtendremos el valor mayor y menor de la lista de números con las funciones **Max** y **Min**

```
Console.Clear();
Console.WriteLine("Funciones Min y Max");

/* Numero Mayor y Menor */
Console.WriteLine("Numero Mayor: {0}", numeros.Max());
Console.WriteLine("Numero Menor: {0}", numeros.Min());
```

- c) Para poder aplicar estos métodos sobre un subconjunto de la lista, primero se deben obtener los elementos filtrados, para luego aplicar la función respectiva.

```
/* Numero Par Mayor y Menor */
Console.WriteLine("Numero Par Mayor: {0}",
    numeros.Where(n => n % 2 == 0).Select(n => n).Max());

Console.WriteLine("Numero Par Menor: {0}",
    numeros.Where(n => n % 2 == 0).Select(n => n).Min());
```

- d) Para el caso de los textos es posible obtener el largo mayor y menor con las funciones **Max** y **Min**, ya que la expresión de consulta trabajaría sobre los largos numéricos.

```
/* Texto Mayor y Menor */
Console.WriteLine("Textos largo mayor : {0}", textos.Max(t => t.Length));
Console.WriteLine("Textos largo menor : {0}", textos.Min(t => t.Length));
```

- e) En el caso de querer obtener los textos que coinciden con el largo menor o mayor, se debe trabajar con una variable auxiliar o una subconsulta.

```
/* Texto mas corto */
Console.WriteLine("Textos mas cortos ");
int menor = textos.Min(t => t.Length);
foreach (string item in textos.Where(t => t.Length == menor))
{
    Console.Write("{0} ", item);
}
Console.WriteLine();
```

Función agregada para el promedio Average

- a) Para la opción 5 agregamos un método de nombre **EjemploAverage()**.
- b) Podemos obtener el promedio de los números de manera directa con el método **Average**

```
Console.Clear();
Console.WriteLine("Funcion Average");

/* Promedio numérico */
Console.WriteLine("Promedio de los números: {0}", numeros.Average());
```

- c) Para obtener el promedio de caracteres de los textos, es necesario trabajar sobre el conjunto de los largos individuales de cada texto como expresión del método **Average**

```
/* Promedio de caracteres */
Console.WriteLine("Promedio de caracteres: {0}", textos.Average(t => t.Length));
```

- d) Al igual que antes, si deseamos obtener el detalle de los textos promedio, debemos trabajar con un auxiliar o una subconsulta.

```
/* Texto sobre el Promedio */
Console.WriteLine("Texto sobre el promedio");
foreach (string item in textos.Where
    (t => t.Length >= textos.Average(t2 => t2.Length)))
{
    Console.WriteLine("{0} ", item);
}
Console.WriteLine();
```