El empleo es de todos

Mintrabajo

APLICAR PROGRAMACIÓN ORIENTADA A OBJETO CON C#

José Fernando Galindo Suárez

SENA

PROGRAMACIÓN ORIENTADA A OBJETO CON C#

5 —

Language INtegrated Query
CON C#

OBJETIVOS

Después de completar esta lección usted estará en la capacidad de:

- **Entender los conceptos básicos de la utilización de Language IN tegrated Query con C#.**
- **Aplicar LINQ en C#**

¿QUÉ ES LINQ?

Es un conjunto de extensiones integradas en el lenguaje C#, que permite trabajar de manera cómoda y rápida con colecciones de datos, como si de una base de datos se tratase.

Llevando a cabo inserciones, selecciones y borrados, así como operaciones sobre sus elementos.

LINQ

LINQ - OPERADORES

OPERADOR	DESCRIPCIÓN
Select SelectMany	Operadores de selección para filtrar y obtener datos.
Where OfType	Operadores de restricción utilizados para filtrar y obtener datos.
OrderBy OrderByDescending Reverse ThenBy ThenByDescending	Operadores de ordenación utilizados para ordenar el resultado de los datos obtenidos.
GroupBy	Operadores de agrupación, para agrupar la salida de los datos obtenidos.
Distinct Except Intersect Union	Operadores de trabajo con listas de datos.

LINQ - OPERADORES

OPERADOR	DESCRIPCIÓN
All Any Contains	Operadores de cuantificación utilizados para obtener los datos de acuerdo a una determinada condición.
GroupJoin Join	Operadores de intersección utilizados para relacionar los elementos de listas de datos.
Skip SkipWhile Take TakeWhile	Operadores de extracción de una cantidad determinada de elementos. Útil por ejemplo para paginaciones de datos.
Aggregate Average Count LongCount Max Min Sum	Operadores de agregación utilizados para obtener resultados de acuerdo a la agregación de información o datos.

LINQ - OPERADORES

OPERADOR	DESCRIPCIÓN
AsEnumerable AsQueryable Cast ToArray ToDictionary ToList ToLookup	Operadores utilizados para transformar la salida de los datos.
ElementAt ElementAtOrDefault First FirstOrDefault Last LastOrDefault Single SingleOrDefault	Operadores para acceder al elemento situado en una determinada posición.
Concat	Operadores de concatenación utilizados para unir datos de un par de listas en una sola lista de datos.
SequenceEqual	Operadores de restricción utilizados para filtrar y o btener datos.

LINQ - OPERADORES

OPERADOR	DESCRIPCIÓN
DefaultIfEmpty **Empty** **Range** **Repeat**	**Operadores de generación para generar un conjunto de datos.**

https://learntutorials.net/es/csharp/topic/68/consultas-linq

LINQ - MÉTODOS

- Aggregate (func)
- gregado (semilla, func)
- Aggregate (seed, func, resultSelector)
- Todo (predicado)
- Any ()
- Any (predicado)
- AsEnumerable ()
- Average ()
- Average (selector)
- Cast <Result> ()
- Concat (segundo)
- Contains (valor)
- Contains (valor, comparador)
- Cuenta ()
- Cuenta (predicado)
- DefaultIfEmpty ()
- DefaultIfEmpty (defaultValue)
- Distinto ()
- Distinto (comparador)
- ElementAt (index)
- ElementAtOrDefault (índice)
- Empty ()
- Excepto (segundo)
- Excepto (segundo, comparador)
- Primera ()
- Primero (predicado)
- FirstOrDefault ()
- FirstOrDefault (predicado)
- GroupBy (keySelector)
- GroupBy (keySelector, resultSelector)
- GroupBy (keySelector, elementSelector)
- GroupBy (keySelector, comparer)
- GroupBy (keySelector, resultSelector, comparer)
- GroupBy (keySelector, elementSelector, resultSelector)
- GroupBy (keySelector, elementSelector, comparer)
- GroupBy (keySelector, elementSelector, resultSelector, comparer)
- Intersect (segundo)
- Intersect (segundo, comparador)

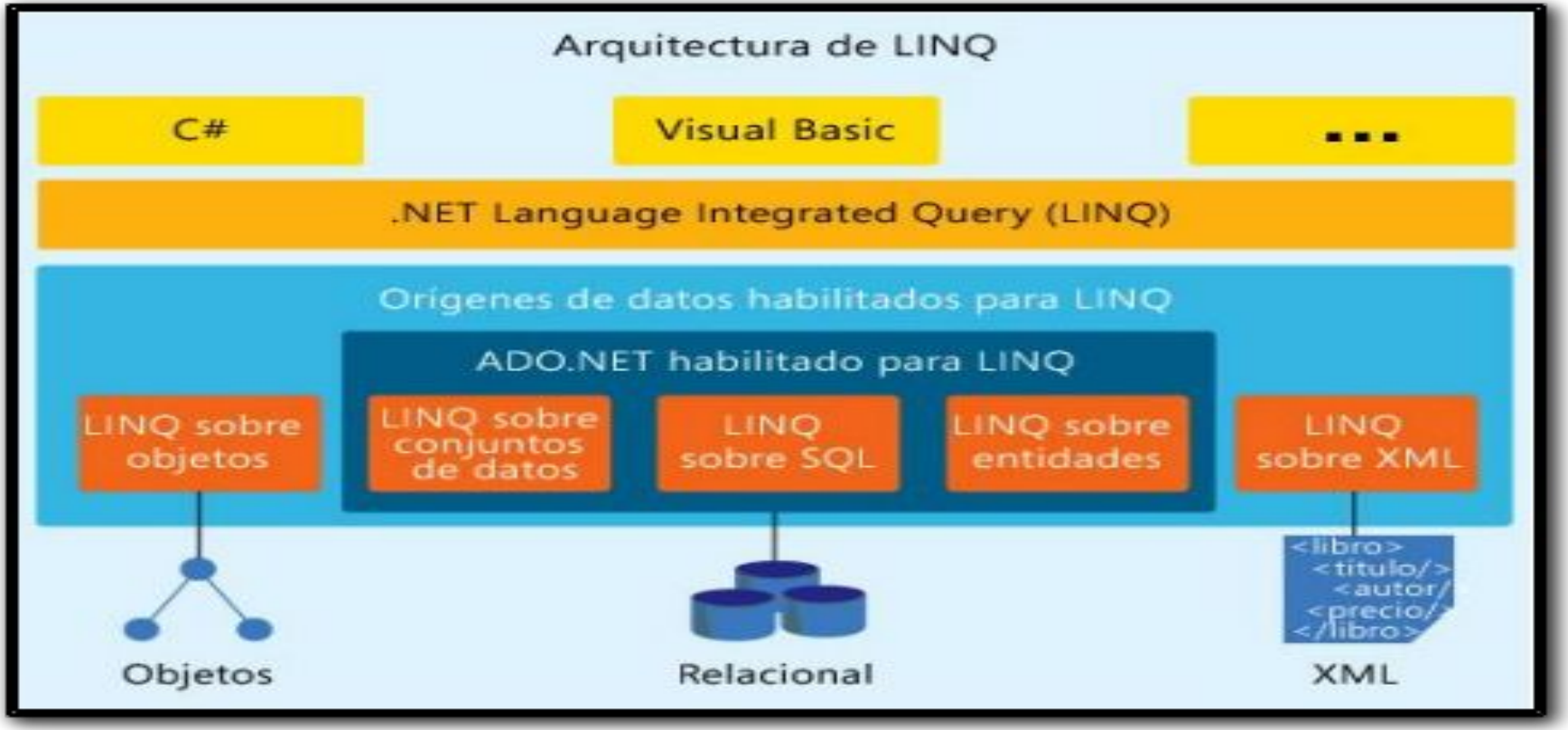
LINQ - MÉTODOS

- Join (inner, outerKeySelector, innerKeySelector, resultSelector)
- Join (inner, outerKeySelector, innerKeySelector, resultSelector, comparer)
- Last ()
- Last (predicado)
- LastOrDefault ()
- LastOrDefault (predicado)
- LongCount ()
- LongCount (predicado)
- Max ()
- Max (selector)
- Min ()
- Min (selector)
- OfType <TResult> ()
- OrderBy (keySelector)
- OrderBy (keySelector, comparer)
- OrderByDescending (keySelector)
- OrderByDescending (keySelector, comparer)
- Rango (inicio, conteo)
- Repetir (elemento, contar)
- Reverse ()
- Seleccionar (selector)
- SelectMany (selector)
- SelectMany (collectionSelector, resultSelector)
- SequenceEqual (segundo)
- SequenceEqual (segundo, comparador)
- Single ()
- Single (predicado)
- SingleOrDefault ()
- SingleOrDefault (predicado)
- Skip (contar)
- SkipWhile (predicado)
- Sum ()
- Sum (selector)
- Tomar (contar)
- TakeWhile (predicado)
- orderThenBy (keySelector)
- orderThenBy (keySelector, comparer)
- orderThenByDescending (keySelector)

LINQ - MÉTODOS

- orderThenByDescending (keySelector, comparer)
- ToArray ()
- ToDictionary (keySelector)
- ToDictionary (keySelector, elementSelector)
- ToDictionary (keySelector, comparer)
- ToDictionary (keySelector, elementSelector, comparer)
- ToList ()
- ToLookup (keySelector)
- ToLookup (keySelector, elementSelector)
- ToLookup (keySelector, comparer)
- ToLookup (keySelector, elementSelector, comparer)
- Union (segundo)
- Union (segundo, comparador)
- Donde (predicado)
- Zip (segundo, resultSelector)

LINQ - ADO.NET



PRÁCTICA EN EL AMBIENTE DE APRENDIZAJE

LINQ - MÉTODOS CAMBIAR, ORDENAR Y FILTRO

```
using System;
using System.Linq;
namespace miLinq1
{
    class Program
    {
        public static void Main(string[] args)
        {
            int[] someNumbers = { 4, 3, 2, 1 };

            var processed = someNumbers
            .Select(n => n * 2)    // Multiplica cada numero por 2
            .Where(n => n != 6)    // except0 para el 6
            .OrderBy(n => n);      // ordenando ascendentemente
            foreach(int x in processed)
                Console.Write("[{0}]",x);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS RANGE

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        public static void Main(string[] args)
        {
            IEnumerable<int> squares = Enumerable.Range(1, 10).Select(x => x * x);
            foreach(int x in squares)
                Console.Write("[{0}]",x);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS SKIP, TAKE

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        static void Muestre(IEnumerable Cual){ 6
            foreach(int x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            var values = new [] { 5, 4, 3, 2, 1 }; 1

2           var skipTwo         = values.Skip(2);          // { 3, 2, 1 }
            var takeThree       = values.Take(3);  3       // { 5, 4, 3 }
4           var skipOneTakeTwo  = values.Skip(1).Take(2); // { 4, 3 }
            var takeZero        = values.Take(0);  5       // Cero elementos
            Muestre( values);
            Muestre( skipTwo);
7           Muestre( takeThree);
            Muestre( takeZero);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS SPLIT, SELECTMANY

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        static void Muestre(IEnumerable Cual){ 3
            foreach(string x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
        1   var palabras=new []{"Amarillo,Azul,Rojo","Cafe,Negro","Blanco"};
            var partayCombine = palabras.SelectMany(x => x.Split(',')); 2
         4  Muestre(partayCombine); //[Amarillo][Azul][Rojo][Cafe][Negro][Blanco]
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS ALL

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        static void Muestre(IEnumerable Cual){ 1
            foreach(int x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            var numbers = new List<int>(){ 1, 2, 3, 4, 5};
        2   Muestre(numbers);
            bool result = numbers.All(i => i < 10); // true 3
            Console.WriteLine("Todos los elementos son menores que 10={0}",result);
        4   result = numbers.All(i => i >= 3); // false
            Console.WriteLine("Todos los elementos son mayores o igual que 3={0}",result);
            //Comprueba si la lista esta vacia
            var numeros = new List<int>();
            result = numeros.All(i => i >= 0); // true
        5   Console.WriteLine("La lista esta vacia={0}",result);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS OFTYPE

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    interface IFoo { }
    class Foo : IFoo { }
    class Bar : IFoo { }
    class Program
    {
        static void Muestre(IEnumerable Cual){
            foreach(int x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            var item0 = new Foo();
            var item1 = new Foo();
            var item2 = new Bar();
            var item3 = new Bar();
            var collection = new IFoo[] { item0, item1, item2, item3 };
            var foos = collection.OfType<Foo>(); // result: IEnumerable<Foo> with item0 and item1
            var bars = collection.OfType<Bar>(); // result: IEnumerable<Bar> item item2 and item3
            var foosAndBars = collection.OfType<IFoo>(); // result: IEnumerable<IFoo> with all four items
            Console.WriteLine("{0}\n{1}\n{2}\n",foos,bars,foosAndBars);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS UNION

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    interface IFoo { }
    class Foo : IFoo { }
    class Bar : IFoo { }
    class Program
    {
        static void Muestre(IEnumerable Cual){ 3
            foreach(int x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            int[] numbers1 = { 1, 2, 3 }; 1
            int[] numbers2 = { 2, 3, 4, 5 };
            var allElement = numbers1.Union(numbers2); 2
            Muestre(allElement); 4
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS DISTINCT

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        static void Muestre(IEnumerable Cual){ 1
            foreach(object x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            int[] array = { 1, 2, 3, 4, 2, 5, 3, 1, 2 }; 2
3           Muestre(array);
            var distinto = array.Distinct(); 4
5           Muestre(distinto);
            Console.WriteLine("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS SELECT WHERE

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Persona{
        public string FirstName{get;set;}
        public string LastName{get;set;}
        public int Age{get;set;}
    }
    class Program
    {

        static void Muestre(IEnumerable Cual){ 1
            foreach(var x in Cual)
                Console.WriteLine("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            Persona[] people ={
2           new Persona { FirstName = "Rosa", LastName = "Melo", Age = 30},
            new Persona { FirstName = "Simon" , LastName = "Tolomeo", Age = 28},
            new Persona{ FirstName = "Aky" , LastName = "Toy", Age = 29},
            new Persona { FirstName = "Zoila" , LastName = "Vaca", Age = 15}
        };
3           var personas = from s in people
                where s.Age <= 28
                select s.FirstName+";"+s.LastName+";"+s.Age;

4           Muestre(personas);
            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS AVERAGE

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{

    class Program
    {

        static void Muestre(IEnumerable Cual){ 1
            foreach(var x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
            int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        2   Muestre(numbers);
            Console.WriteLine("El promedio de la lista es {0}",numbers.Average());
            Console.WriteLine("La sumatoria de la lista es {0}",numbers.Sum());
        3   Console.WriteLine("La Cantidad de elementos de la lista es {0}",numbers.Count());
            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

LINQ - MÉTODOS CONTAINS

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Collections;
namespace miLinq1
{
    class Program
    {
        static void Muestre(IEnumerable Cual){
            foreach(int x in Cual)
                Console.Write("[{0}]",x);
            Console.WriteLine("\n*****************************");
        }
        public static void Main(string[] args)
        {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
        Muestre(numbers);
        var result1 = numbers.Contains(4); // true
        Console.WriteLine("La lista contiene un 4 {0}",result1);
        var result2 = numbers.Contains(8); // false
        Console.WriteLine("La lista contiene un 8 {0}",result2);
        List<int> secondNumberCollection = new List<int> { 4, 5, 6, 7 };
        Muestre(secondNumberCollection);
        Console.WriteLine("Cuales numeros estan en ambas listas (numbers y secondNumberCollection)");
        var result3 = secondNumberCollection.Where(item => numbers.Contains(item)); // { 4,5}
        Muestre(result3);
        Console.WriteLine("Oprima una tecla....");
        Console.ReadLine();
        }
    }
}
```

1
2
3

LINQ

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace miLinq
{
    public class Alumno
    {
    public string Nombre { get; set; }
    public int Nota { get; set; }
    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace miLinq
{
    class Program
    {
        public static void Main(string[] args)
        {
             var alumnos = new List<Alumno>{
                new Alumno {Nombre = "Pedro",Nota = 5},
                new Alumno {Nombre = "Jorge",Nota = 8},
                new Alumno {Nombre = "Andres",Nota = 3}
                };

        foreach(Alumno x in alumnos)
            Console.WriteLine("{0} saco una nota de {1}",x.Nombre,x.Nota);

        Console.WriteLine("Primero->{0}",alumnos.First().Nombre);
        Console.WriteLine("Ultimo->{0}",alumnos.Last().Nombre);

        var resultado = from alumno in alumnos
                where alumno.Nota >= 5
                orderby alumno.Nota
                select alumno;
        Console.WriteLine("*****************************************************");
        foreach(Alumno x in resultado)
            Console.WriteLine("{0} saco una nota de {1}",x.Nombre,x.Nota);
        Console.WriteLine("Digite una tecla.....");
Console.ReadKey(true);
        }
    }
}
```

1 2 3 4 5 6

EVIDENCIA DE APRENDIZAJE

EVIDENCIA DE APRENDIZAJE

Desarrollar los códigos de programación propuestos, ejecutarlos y entregarlos al instructor subiendo cada salida y el código a TERRITORIUM.

<u>Aprendiz</u>, No olvide subirla a su portafolio de evidencias.

DESCARGAR ARCHIVOS

COMPILADOR SHARPDEV

VERSIÓN PDF

CRÉDITOS

Realizado por el instructor José Fernando Galindo Suárez
jgalindos@sena.edu.co 2022

G R A C I A S

Línea de atención al ciudadano: 018000 910270
Línea de atención al empresario: 018000 910682

@SENAcomunica

www.sena.edu.co