**Expresiones Lambda en C#**

1. **¿Qué son las expresiones lambda?**

Las expresiones lambda son una forma concisa de escribir funciones anónimas en C#. Permiten definir un bloque de código como si fuera un método, pero sin necesidad de declararlo explícitamente con nombre y firma.

Su sintaxis general es:

(parametros) => expresion\_o\_bloque

1. **Ventajas de las expresiones lambda**

* Código más conciso y legible.
* Permiten definir comportamiento en línea sin crear un método aparte.
* Muy útiles para delegados de una sola línea, eventos o callbacks rápidos.
* Se integran de forma natural con LINQ y colecciones genéricas.

Ejemplos:

using System;

class Program

{

static void Main(string[] args)

{

// Func<int,int> -> recibe un int y devuelve un int

Func<int, int> square = x => x \* x;

int number = 7;

Console.WriteLine($"El cuadrado de {number} es: {square(number)}");

// Func<DateTime> -> no recibe nada y devuelve un DateTime

Func<DateTime> getNow = () => DateTime.Now;

Console.WriteLine($"La fecha y hora actual es: {getNow()}");

// Action<string> -> recibe un string y no devuelve nada

Action<string> printMessage = msg => Console.WriteLine(msg);

printMessage("Hola desde una expresión lambda con Action<string>!");

}

}

Otro ejemplo:

class Program

{

static void Main(string[] args)

{

List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

List<int> listPairs = numbers.FindAll(n => n % 2 == 0);

foreach (int number in listPairs)

{

Console.WriteLine(number);

}

}

}

1. **Consideraciones de sintaxis para expresiones lambda en C#**
   1. **Número de parámetros y paréntesis**
   * Un solo parámetro: Los paréntesis son opcionales.
   * x => x \* x; // Correcto
   * (x) => x \* x; // También correcto
   * Cero parámetros: Los paréntesis son obligatorios.
   * () => DateTime.Now;
   * Múltiples parámetros: Los paréntesis son obligatorios.
   * (a, b) => a + b;
   1. **Cuerpo de la lambda**
   * Expresión única (expression-bodied): No necesita llaves {} ni return.
   * x => x \* x;
   * Bloque de sentencias (statement-bodied): Si el cuerpo tiene más de una instrucción, se requieren llaves {} y return (cuando hay valor de retorno).
   * (x, y) =>

{

int suma = x + y;

return suma;

};

* 1. **Inferencia de tipos**
  + El compilador infiera automáticamente los tipos de los parámetros según el contexto (Func<>, Action<>, delegate).
  + Si se desea, se pueden especificar explícitamente:
  + (int x, int y) => x + y;
  1. **Ámbito (scope) y variables capturadas**
  + Una lambda puede capturar variables locales del método donde se declara. Esto se llama closures.

using System;

class Program

{

static void Main(string[] args)

{

int factor = 10; // variable local a capturar por la lambda

// La lambda captura la variable 'factor'

Func<int, int> multiplicar = n => n \* factor;

Console.WriteLine("Usando closure:");

Console.WriteLine(multiplicar(5)); // 5 \* 10 = 50

// Si la variable cambia, la lambda "recuerda" la referencia, no solo el valor

factor = 20;

Console.WriteLine(multiplicar(5)); // 5 \* 20 = 100

// Ejemplo adicional: devolver una lambda que sigue recordando el 'factor'

Func<int, int> CrearMultiplicador(int f)

{

return n => n \* f; // 'f' queda capturado por el closure

}

var multiplicarPor3 = CrearMultiplicador(3);

var multiplicarPor7 = CrearMultiplicador(7);

Console.WriteLine(multiplicarPor3(4)); // 12

Console.WriteLine(multiplicarPor7(4)); // 28

}

}

* + Consideración: la variable capturada se mantiene viva mientras la lambda exista.
  1. **Retorno**
  + Si es una expresión única, el valor retornado es implícito.
  + En un bloque, se requiere return.
  1. **Resumen rápido de los paréntesis en lambdas:**
  + 0 parámetros → () => ...
  + 1 parámetro → x => ... (sin paréntesis) o (x) => ... (con paréntesis, opcional)
  + +1 parámetro → (x, y, z) => ...