María Luz Fernández Gismero.

UD2: Características avanzadas de C#Transformación digital

1. Introducción

En la primera unidad aprendimos los fundamentos básicos de C#: tipos de datos, estructuras de control, métodos y programación orientada a objetos.
En esta unidad vamos a dar un paso más y estudiar **características avanzadas del lenguaje** que nos permitirán escribir programas más flexibles, reutilizables y fáciles de mantener.

Estas características incluyen:

- **Colecciones**, que nos permiten manejar conjuntos de datos de manera dinámica.
- **Delegados y eventos**, base de la programación orientada a eventos.
- **Expresiones lambda**, una forma más concisa de escribir funciones.
- **LINQ**, un sistema de consultas integrado en el propio lenguaje.
- **Genéricos**, que permiten escribir código adaptable a cualquier tipo de dato.

El objetivo es adquirir una visión práctica de estas herramientas y aprender a utilizarlas en situaciones reales.

2. Colecciones en C#

2.1. Concepto

En programación, no siempre trabajamos con un único valor. Muchas veces necesitamos manejar **conjuntos de elementos** relacionados: la lista de alumnos de una clase, los productos de una tienda o los clientes en una cola de espera.

En C# existen los **arrays**, que permiten almacenar varios valores, pero tienen un **tamaño fijo** y funcionalidades limitadas. Por eso, el lenguaje ofrece las **colecciones**, que son más potentes y dinámicas.

Las colecciones permiten:

- Agregar y eliminar elementos en cualquier momento.

- Recorrer los elementos de forma sencilla.
- Realizar búsquedas y ordenaciones con facilidad.

2.2. Arrays

Un **array** es una estructura que almacena un número fijo de elementos del mismo tipo. Se crea indicando el tipo de dato y el tamaño.

Ejemplo:

```
int[] numeros = new int[5];   // Array de 5 enteros

numeros[0] = 10;

numeros[1] = 20;
```

Los arrays se recorren con bucles for o foreach.

2.3. List

Una **List<T>** es una colección genérica que permite almacenar elementos de un mismo tipo.
A diferencia de los arrays, su tamaño puede **crecer o reducirse dinámicamente**.

Características:

- No requiere definir tamaño inicial.
- Permite añadir, eliminar, ordenar y buscar.
- Muy utilizada en programas de todo tipo.

Ejemplo:

```
List<string> nombres = new List<string>();

nombres.Add("Ana");

nombres.Add("Luis");
```

2.4. Dictionary

Un **Dictionary<TKey, TValue>** es una colección que almacena datos en forma de **pares clave–valor**.
Cada clave es única y se usa para acceder rápidamente al valor asociado.

Características:

- Acceso rápido a los datos mediante la clave.
- Útil para representar relaciones: matrícula → alumno, DNI → persona.

Ejemplo:

```
Dictionary<string, int> edades = new Dictionary<string, int>();
```

edades["Ana"] = 20;

edades["Luis"] = 22;

2.5. Queue

Una **Queue<T>** o cola es una colección que sigue el principio **FIFO** (*First In, First Out*): el primer elemento en entrar es el primero en salir.

Características:

- Garantiza el orden de llegada.
- Muy utilizada en simulaciones de colas o en tareas en segundo plano.

Ejemplo:

Queue<string> clientes = new Queue<string>();

clientes.Enqueue("Carlos");

clientes.Enqueue("María");

string siguiente = clientes.Dequeue(); // "Carlos"

3. Delegados y eventos

3.1. Delegados

Un **delegado** es un tipo especial que almacena la referencia a un método.
En otras palabras, es como una "variable que apunta a una función".
Gracias a los delegados, podemos pasar métodos como parámetros y cambiar el comportamiento de un programa de forma dinámica.

Ejemplo sencillo:

delegate void Saludo(string nombre);

void DecirHola(string n) => Console.WriteLine("Hola " + n);

Saludo s = DecirHola;

s("Ana"); // Invoca DecirHola

3.2. Eventos

Los **eventos** permiten que un objeto **notifique** a otros objetos cuando ocurre algo.
Se basan en delegados, pero añaden un mecanismo de seguridad.

Ejemplo típico: un botón que lanza un evento Click cuando el usuario lo pulsa.
En nuestras aplicaciones de consola, podemos simular un evento cuando el stock de un producto baja demasiado.

4. Expresiones lambda

Las **lambdas** son funciones anónimas, es decir, funciones sin nombre, que se escriben de forma muy concisa.
Se utilizan principalmente con delegados, eventos y LINQ.

Sintaxis:

(parámetros) => expresión

Ejemplo:

Func<int, int> cuadrado = x => x * x;

Console.WriteLine(cuadrado(5)); // 25

Ventajas:

- Código más corto y claro.
- Muy útil para definir funciones "rápidas" sin necesidad de escribir métodos completos.

5. LINQ sobre colecciones

5.1. Concepto

LINQ (Language Integrated Query) permite hacer **consultas a colecciones** de datos directamente desde C#.
La sintaxis recuerda a SQL, pero está integrada en el lenguaje.

Ejemplo:

List<int> numeros = new List<int> { 1, 2, 3, 4, 5, 6 };

var pares = numeros.Where(n => n % 2 == 0);

5.2. Operaciones comunes

- **Where**: filtrar elementos.
- **Select**: proyectar (transformar) elementos.
- **OrderBy**: ordenar.
- **Sum, Count, Average**: operaciones de agregación.

LINQ funciona con cualquier colección que implemente IEnumerable<T>.

6. Genéricos

6.1. Concepto

Los **genéricos** permiten crear clases, interfaces y métodos que funcionan con cualquier tipo de dato.
En lugar de duplicar código, escribimos una plantilla que se adapta al tipo que necesitamos.

Ejemplo con lista:

List<int> listaEnteros = new List<int>();

List<string> listaCadenas = new List<string>();

Ambas listas utilizan la misma clase genérica List<T>.

6.2. Clase genérica personalizada

Podemos crear nuestras propias clases genéricas:

class Caja<T>

{

public T Contenido { get; set; }

}

Esto permite instanciar Caja<int> o Caja<string> con la misma definición.

6.3. Métodos genéricos

También podemos crear **métodos genéricos**:

static T DevolverPrimero<T>(List<T> lista)

{

return lista[0];

}

De esta forma, el método funciona con cualquier tipo.

7. Conclusión

Con las características avanzadas de C#, pasamos de programas básicos a aplicaciones mucho más potentes y flexibles.

- Las **colecciones** permiten organizar los datos de manera eficiente.
- Los **delegados y eventos** introducen un modelo de programación orientado a sucesos.
- Las **expresiones lambda** hacen que el código sea más conciso y expresivo.
- Con **LINQ** podemos realizar consultas complejas con poco código.

- Los **genéricos** nos permiten escribir estructuras y métodos reutilizables para cualquier tipo de dato.

Dominar estos conceptos es fundamental para dar el salto hacia aplicaciones profesionales, como las que construiremos en las próximas unidades con **ASP.NET Core**.