Fundamentos de Pilas (Stack<T>) en C#

1. ¿Qué es una pila?

En C#, una pila es una colección que almacena elementos siguiendo la estrategia LIFO (Last In, First Out), es decir, el último elemento que entra es el primero que sale.
Esto es útil en escenarios como historial de acciones, operaciones de deshacer/rehacer, evaluación de expresiones, navegación en menús, entre otros.
La clase que se usa para pilas genéricas es Stack<T>, donde T representa el tipo de datos que contendrá (por ejemplo, Stack<int>, Stack<string>, Stack<Operacion>).

Características principales:

- Se encuentra en el namespace System.Collections.Generic.
- El acceso se realiza desde el tope de la pila (último elemento agregado).
- Métodos principales:
  - **Push** → Agrega un elemento en la parte superior.
  - **Pop** → Elimina y devuelve el elemento en la parte superior.
  - **Peek** → Devuelve el elemento en la parte superior sin eliminarlo.
- Crece o reduce su tamaño dinámicamente.

2. Tipos de colecciones y la posición de Stack<T>

En el ecosistema de colecciones de C#, tenemos:

- Array → tamaño fijo, acceso aleatorio rápido.
- List → tamaño dinámico, acceso por índice.
- Dictionary<TKey, TValue> → pares clave-valor.
- Queue → estructura FIFO (First In, First Out).
- Stack → estructura LIFO (Last In, First Out).

En general:

- Queue es como una fila de espera (sale el primero que entra).
- Stack es como una pila de platos (se retira el último que se puso).

3. Creación de una pila

Antes de usar pilas, se debe incluir el espacio de nombres:

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

Formas de crear una pila:

```
// Pila vacía
Stack<string> operaciones = new Stack<string>();
```

```
// Pila con datos iniciales
Stack<int> numeros = new Stack<int>(new[] { 1, 2, 3 });
```

---

4. Operaciones básicas con Stack<T>

4.1. Agregar elementos (Push)

```
operaciones.Push("Abrir archivo");
operaciones.Push("Escribir texto");
```

4.2. Consultar el elemento superior sin eliminar (Peek)

```
string ultima = operaciones.Peek();
Console.WriteLine(ultima);
```

4.3. Eliminar el elemento superior y devolverlo (Pop)

```
string deshecha = operaciones.Pop();
Console.WriteLine($"Deshechando: {deshecha}");
```

4.4. Contar elementos (Count)

```
Console.WriteLine(operaciones.Count);
```

5. Ejemplo: Gestor de Deshacer (UndoManager)

Este ejemplo simula un editor donde se realizan operaciones y se mantiene un historial para poder deshacerlas.

```
public class UndoManager
{
    public static void Main(string[] args)
    {
        Stack<string> operationHistory = new Stack<string>();

        Console.WriteLine("\nEscribiendo Hola Mundo");
        operationHistory.Push("1. Escribir texto 'Hola Mundo'");
        Thread.Sleep(1000);

        Console.WriteLine("Escribiendo texto \"Hola Mundo\"");
        operationHistory.Push("2. Texto escrito \"Hola Mundo\"");
        Thread.Sleep(1000);

        Console.WriteLine("Dibujando círculo rojo");
        operationHistory.Push("3. Círculo dibujado en rojo");
        Thread.Sleep(1000);

        Console.WriteLine("Poniendo título en negrita");
        operationHistory.Push("4. Poner en negrita un título");
        Thread.Sleep(1000);

        Console.WriteLine($"\nOperaciones en el historial:
{operationHistory.Count}");

        if (operationHistory.Count > 0)
        {
```

```
            string lastOperation = operationHistory.Peek();
            Console.WriteLine($"\nÚltima operación: '{lastOperation}'
(aún no deshecha)");
        }

        Console.WriteLine("\nDeshaciendo operaciones:");
        while (operationHistory.Count > 0)
        {
            string undoneOperation = operationHistory.Pop();
            Console.WriteLine($"  - Deshaciendo: '{undoneOperation}'");
            Thread.Sleep(1000);
        }

        Console.WriteLine("\nTodas las operaciones han sido deshechas.");
        Console.WriteLine($"Operaciones restantes:
{operationHistory.Count}");
        Console.ReadKey();
    }
}
```

6. Evolución de la pila en memoria

Punto de partida

```
Stack<string> historial = new Stack<string>();
// historial → [ ]  (Count = 0)
```

Después de Push("A"), Push("B"), Push("C")

```
Top → [ "C", "B", "A" ] (Count = 3)
```

Peek() devuelve "C" sin eliminarlo.

Pop() elimina "C":

```
Top → [ "B", "A" ] (Count = 2)
```

7. 7. Métodos útiles de Stack<T>

Método	Descripción
Push(item)	Inserta un elemento en la parte superior.
Pop()	Elimina y devuelve el elemento superior.
Peek()	Devuelve el elemento superior sin eliminarlo.
Count	Devuelve la cantidad de elementos en la pila.
Clear()	Elimina todos los elementos.
Contains(item)	Verifica si el elemento existe.
ToArray()	Convierte la pila a un arreglo.

8. Buenas prácticas

- Usar Stack<T> cuando el orden LIFO sea el requerido.
- Evitar usar Pop() sin verificar Count > 0 para prevenir excepciones.
- Si necesitas recorrer sin modificar la pila, usar foreach (no cambia el orden interno).
- En sistemas de deshacer/rehacer, usar dos pilas: una para “Undo” y otra para “Redo”.