МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Лабораторна робота №1

з дисципліни «Розробка програмного забезпечення на платформі .Net»

на тему:

«Узагальнені типи (Generic) з підтримкою подій. Колекції»

Виконав:

Студент групи ІС-12

Канупа Максим

Викладачк:
Бардін В.

Київ – 2023

Мета лабораторної роботи – навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

Завдання:

1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та ін. (відповідно до типу колекції).
2. Додати до класу власної узагальненої колекції підтримку подій та обробку виключних ситуацій.
3. Опис класу колекції та всіх необхідних для роботи з колекцією типів зберегти у динамічній бібліотеці.
4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

Варіант:

7	Відсортований динамічний масив	Див. SortedList<T>	Збереження даних за допомогою динамічно зв'язаного списку або вектору

Реалізація:
SortedList.cs

```
using System.Collections;

namespace Lab1.SortedList;

public class SortedList<T> : ICollection<T> where T : IComparable<T>
{
    private Node<T>? _head;
    public int Count { get; private set; }
    public bool IsReadOnly => false;

    public event EventHandler? OnAdd;
    public event EventHandler? OnRemove;
    public event EventHandler? OnClear;

    public IEnumerator<T> GetEnumerator()
    {
        var node = _head;
        while (node is not null)
        {
            yield return node.Value;
            node = node.Next;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    public void Add(T item)
    {
        ArgumentNullException.ThrowIfNull(item);

        Count++;
        var newNode = new Node<T>
        {
            Value = item
        };

        if (_head is null || item.CompareTo(_head.Value) < 0)
        {
            newNode.Next = _head;
            _head = newNode;
            return;
        }

        var node = _head;
        while (node.Next is not null && item.CompareTo(node.Next.Value) > 0)
        {
            node = node.Next;
        }

        newNode.Next = node.Next;
```

```
        node.Next = newNode;

        OnAdd?.Invoke(this, EventArgs.Empty);
    }

    public T? Find(Predicate<T> match)
    {
        foreach (var item in this)
        {
            if (match(item))
            {
                return item;
            }
        }

        return default;
    }

    public void Clear()
    {
        _head = null;
        Count = 0;

        OnClear?.Invoke(this, EventArgs.Empty);
    }

    public bool Contains(T item)
    {
        ArgumentNullException.ThrowIfNull(item);

        foreach (var node in this)
        {
            if (node.CompareTo(item) == 0)
            {
                return true;
            }
        }

        return false;
    }

    public void CopyTo(T[] array, int arrayIndex)
    {
        ArgumentNullException.ThrowIfNull(array);
        if (arrayIndex < 0)
        {
            throw new ArgumentOutOfRangeException(nameof(arrayIndex));
        }
        if (array.Length - arrayIndex < Count)
        {
            throw new ArgumentException("Destination array is not long enough to
copy all the items in the collection.");
        }

        var i = arrayIndex;
        foreach (var item in this)
        {
```

```
            array[i] = item;
            i++;
        }
    }

    public bool Remove(T item)
    {
        ArgumentNullException.ThrowIfNull(item);

        var node = _head;
        if (node is null) return false;

        if (node.Value.CompareTo(item) == 0)
        {
            Count--;
            _head = node.Next;

            OnRemove?.Invoke(this, EventArgs.Empty);
            return true;
        }

        while (node.Next is not null)
        {
            if (node.Next.Value.CompareTo(item) == 0)
            {
                Count--;
                node.Next = node.Next.Next;

                OnRemove?.Invoke(this, EventArgs.Empty);
                return true;
            }

            node = node.Next;
        }

        return false;
    }

    public override string ToString()
    {
        return string.Join(", ", this);
    }
}
```

Node.cs

```
namespace Lab1.SortedList;

public class Node<T>
{
    public required T Value { get; init; }
    public Node<T>? Next { get; set; }
}
```

Program.cs

```
using Lab1.SortedList;
```

```
var sortedList = new SortedList<int>();

sortedList.OnAdd += (sender, args) => Console.WriteLine("Element added!");
sortedList.OnRemove += (sender, args) => Console.WriteLine("Element removed!");
sortedList.OnClear += (sender, args) => Console.WriteLine("List cleared!");

Console.WriteLine(sortedList.Count);
Console.WriteLine("Add items");
sortedList.Add(2);
sortedList.Add(1);
sortedList.Add(-1);
sortedList.Add(0);
sortedList.Add(0);
sortedList.Add(2);
sortedList.Add(-9);
Console.WriteLine(sortedList.Count);

sortedList.Remove(0);

var arr = new int[sortedList.Count];
sortedList.CopyTo(arr, 0);
Console.WriteLine(string.Join(", ", arr));

Console.WriteLine(sortedList.Count);
Console.WriteLine(sortedList);
Console.WriteLine(sortedList.Remove(-9));
Console.WriteLine(sortedList.Remove(-8));
sortedList.Clear();

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

Результати виконання

```
D:/Code/RiderProjects/Lab1/ConsoleApp/bin/Debug/net7.0/ConsoleApp.exe
0
Add items
Element added!
Element added!
Element added!
7
Element removed!
-9, -1, 0, 1, 2, 2
6
-9, -1, 0, 1, 2, 2
Element removed!
True
False
List cleared!
0
```