Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 1

з дисципліни «Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

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

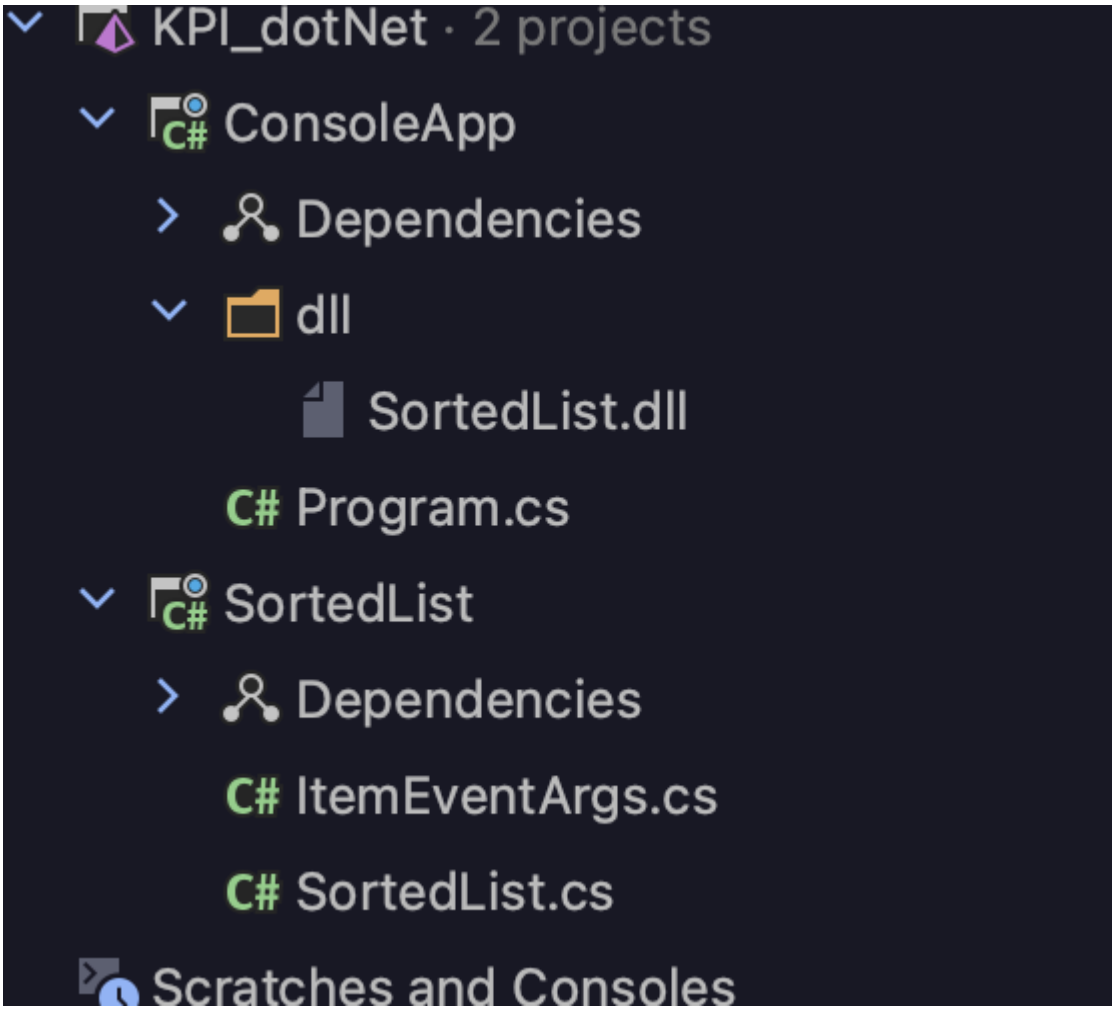
Виконав:
студент групи ІА-13
Прізвище Ім’я.
Дата здачі _____
Захищено з балом _____

Перевірила:
ст. вик. кафедри ІПІ
Крамар Ю. М.

Київ, 2023

Тема: Узагальнені типи (Generic) з підтримкою подій. Колекції
Мета: навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

Код програми:



SortedList.cs

```
using System.Collections;
using SortedList.MyEventArgs;

namespace SortedList;

public class SortedList<T> : ICollection<T> where T : IComparable<T>
{
    private MyNode<T>? _head;
    private MyNode<T>? _tail;
    public bool IsReadOnly { get; } = false;
```

```
    public int Count { get; private set; } // length of the list

    public int Version { get; private set; }


    public event EventHandler<ItemEventArgs<T>>? ItemAdded;

    public event EventHandler<ItemEventArgs<T>>? ItemRemoved;

    public event EventHandler? ListCleared;


    private void InvokeItemAdded(T item) => ItemAdded?.Invoke(this,new
ItemEventArgs<T>(item));


    private void InvokeItemRemoved(T item) =>
ItemRemoved?.Invoke(this,new ItemEventArgs<T>(item));


    private void InvokeListCleared() => ListCleared?.Invoke(this,
EventArgs.Empty);


    public IEnumerator<T> GetEnumerator()

    {

        IEnumerator <T> enumerator = new MyEnumerator(_head, this);

        return enumerator;

    }


    public IEnumerator<T> Reversed()

    {

        var listStarterVersion = Version;

        var current = _tail;

        while (current != null)

        {

            // version check

            if (listStarterVersion != Version)

            {
```

```
                throw new InvalidOperationException("Collection was
modified");
            }

            yield return current.Item;
            current = current.Prev;
        }
    }

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

    public void Add(T item)
    {
        if (item == null)
        {
            throw new ArgumentNullException($"{typeof(T)} {nameof(item)}
is null");
        }

        // head is null
        if (_head == null)
        {
            _head = new MyNode<T>(item);
            _tail = _head;
            InvokeItemAdded(item);
            IncrementCount();
            UpdateVersion();
```

```
            return;
        }
        // item < head
        if (_head.Item.CompareTo(item) > 0)
        {
            var node = new MyNode<T>(item) { Next = _head };
            _head.Prev = node;
            _head = node;
            InvokeItemAdded(item);
            IncrementCount();
            UpdateVersion();

            return;
        }
        // cycles till the end
        var current = _head;
        while (current.Next != null)
        {
            if (current.Next.Item.CompareTo(item) > 0)
            {
                var node = new MyNode<T>(item) { Next = current.Next, Prev
= current};
                current.Next.Prev = node;
                current.Next = node;
                InvokeItemAdded(item);
                IncrementCount();
                UpdateVersion();

                return;
            }
```

```
            current = current.Next;
        }

        _tail = new MyNode<T>(item) { Prev = current };
        current.Next = _tail;
        InvokeItemAdded(item);
        IncrementCount();
        UpdateVersion();
    }

    public void Clear()
    {
        _head = null;
        _tail = null;
        InvokeListCleared();
        ResetVersion();
        ResetCount();
    }

    public bool Contains(T item)
    {
        var current = _head;
        while (current != null)
        {
            var temp = current.Item.CompareTo(item);
            if (temp > 0) return false; // further numbers are bigger //
changed
            if (temp == 0) return true;
            current = current.Next;
        }
```

```
        return false;
    }

    private MyNode<T>? FindNodeByItem(T item)
    {
        var current = _head;
        while (current != null)
        {
            var temp = current.Item.CompareTo(item);
            if (temp > 0) return null; // further numbers are bigger
            if (temp == 0) return current;
            current = current.Next;
        }

        return null;
    }

    public void CopyTo(T[] array, int arrayIndex)
    {
        if (array == null)
        {
            throw new ArgumentNullException($"Array {nameof(array)} is
null");
        }

        if (array.Length - arrayIndex < Count )
        {
            throw new ArgumentException("Not enough space. Count > array
length - starting index");
        }
```

```
        if (arrayIndex < 0 || arrayIndex >= array.Length)
        {
            throw new ArgumentException($"Invalid Argument. arrayIndex = {arrayIndex}. It has to be greater than zero and smaller than array length");
        }

        var i = 0;
        foreach (var item in this)
        {
            array[arrayIndex + i] = item;
            i++;
        }
    }

    public bool Remove(T item)
    {
        if (item == null)
        {
            throw new ArgumentNullException($"{typeof(T)} {nameof(item)} is null");
        }

        var searchedNode = FindNodeByItem(item);
        if (searchedNode == null) return false;

        if (searchedNode == _head && searchedNode == _tail)
        {
            _head = null;
            _tail = null;
            InvokeListCleared();
```

```
            ResetVersion();
            ResetCount();

            return true;
        } if (searchedNode == _head)
        {
            _head = _head.Next;
            _head!.Prev = null;
            InvokeItemRemoved(item);
            DecrementCount();
            UpdateVersion();

            return true;
        } if (searchedNode == _tail)
        {
            _tail = _tail.Prev;
            _tail!.Next = null;
            InvokeItemRemoved(item);
            DecrementCount();
            UpdateVersion();

            return true;
        }

        searchedNode.Next!.Prev = searchedNode.Prev;
        searchedNode.Prev!.Next = searchedNode.Next;
        InvokeItemRemoved(item);
        DecrementCount();
        UpdateVersion();
```

```
        return true;
    }

    // version and length control methods
    private void UpdateVersion() => Version++;

    private void ResetVersion() => Version = 0;

    private void IncrementCount() => Count++;

    private void DecrementCount() => Count--;

    private void ResetCount() => Count = 0;

    // Enumerator
    private class MyEnumerator : IEnumerator<T>
    {
        private MyNode<T>? _current;
        private MyNode<T>? _head;

        private readonly SortedList<T> _list;
        private readonly int _listStarterVersion;

        public MyEnumerator(MyNode<T>? head, SortedList<T> list)
        {
            _current = null;
            _head = head;
            _list = list;
            _listStarterVersion = _list.Version;
        }
```

```
        public T Current
        {
            get
            {
                CheckVersion();
                if (_current == null)
                    throw new InvalidOperationException("Enumeration has
not been started ot it is already finished");
                return _current.Item; // THIS ENSURES THAT USER WILL WORK
WITH T INSTEAD OF THE NODE
            }
        }

        private void CheckVersion()
        {
            if (_listStarterVersion != _list.Version)
            {
                throw new InvalidOperationException("Collection was
modified");
            }
        }

        public bool MoveNext()
        {
            CheckVersion();
            if (_current == null)
            {
                if (_head == null)
                    return false; // list is empty
```

```
                _current = _head;
                return true;
            }

            _current = _current.Next;
            return _current != null;
        }

        public void Reset()
        {
             throw new NotSupportedException("Not implemented due to
safety reasons");
            // If changes are made to the collection,
            // such as adding, modifying, or deleting elements,
            // the behavior of Reset is undefined.
            //
https://learn.microsoft.com/en-us/dotnet/api/system.collections.ienumer
ator.reset?view=net-6.0
        }

        object IEnumerator.Current => Current;

        public void Dispose() { }
    }
    private class MyNode<TU>
    {
        public TU Item { get; set; }
        public MyNode<TU>? Next { get; set; }
        public MyNode<TU>? Prev { get; set; }

        public MyNode(TU value)
```

```
        {
            Item = value;
        }
    }
}
```

ConsoleApp Program.cs

```
using SortedList;

namespace ConsoleApp;

public class Program
{
    public static void Main(string[] args)
    {
        // Initialization
        var list = new SortedList<int>() {1, 2, -5, 11};
        Print(list);
        // Events
        var AddCounter = 0;
        var RemoveCounter = 0;
        list.ItemAdded += (e,s) => AddCounter++;
        list.ItemRemoved += (e,s) => RemoveCounter++;
        list.ListCleared += (e, s) => Console.WriteLine("LIST WAS
CLEARED.....");
        Console.WriteLine($"Add: {AddCounter}  Remove: {RemoveCounter}");
        // Add Remove Methods
        list.Add(33);
        list.Remove(4);
        list.Remove(1);
        Print(list);
```

```
        Console.WriteLine($"Add: {AddCounter}  Remove: {RemoveCounter}");

        // Copy To

        var arr = new int[list.Count + 2];

        arr[0] = 0;

        arr[1] = 0;

        list.CopyTo(arr,2);

        Print(arr);

        // Contains

        Console.WriteLine(list.Contains(1).ToString() + "  " +
list.Contains(33).ToString());

        // Clear Method

        list.Clear();

        Print(list);

    }


    static void Print<T>(IEnumerable<T> list)

    {

        Console.Write("[");

        foreach (var var in list)

        {

            Console.Write(" " + var + ",");

        }


        var temp = Console.GetCursorPosition();

        Console.SetCursorPosition(temp.Left - 1, temp.Top);

        Console.WriteLine("]");

        // Reversed Enumerator

        Console.Write("[");

        foreach (var var in list.Reverse())

        {

            Console.Write(" " + var + ",");
```

```
        }


        temp = Console.GetCursorPosition();
        Console.SetCursorPosition(temp.Left - 1, temp.Top);
        Console.WriteLine("]");
    }
}
```

Результат роботи програми:

```
[ -5, 1, 2, 11]
[ 11, 2, 1, -5]
Add: 0  Remove: 0
[ -5, 2, 11, 33]
[ 33, 11, 2, -5]
Add: 1  Remove: 1
[ 0, 0, -5, 2, 11, 33]
[ 33, 11, 2, -5, 0, 0]
False  True
LIST WAS CLEARED.....
]
]
```

Висновок: виконоючи лабораторну роботу я познайомився з Узагальненими

типами та подіями та колекціями. У ході лабораторної роботи я написав свою колекція з використанням узагальнених типів та підтримкою подій