МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №1

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

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

Викладач:

Бардін В.

Виконав студент:

Лазюта Олексій

групи ІП-15 ФІОТ

Київ-2023

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 - 2

2	Черга	Див. Queue<T>	Збереження даних за допомогою динамічно зв'язаного списку

Код програми:

Файл Node.cs:

```
namespace MyLibrary;

public class Node<T>
{
    public Node<T>? Next { get; set; }
    public readonly T Value;

    public Node(T value)
    {
        Next = null;
        this.Value = value;
    }
}
```

Файл CustomQueue.cs:

```
using System.Collections;

namespace MyLibrary;

public class CustomQueue <T>: IEnumerable<T> where T: IComparable
{
    private int _size;
    private Node<T>? _headNode;
    private Node<T>? _tailNode;

    public event EventHandler? OnEnqueued;
    public event EventHandler? OnDequeued;
    public event EventHandler? OnCleared;
    public event EventHandler? OnCopiedTo;
    public event EventHandler? OnReversed;

    public int Count => _size;

    public CustomQueue()
    {
        _headNode = null;
        _tailNode = null;
        _size= 0;
    }

    public CustomQueue(IEnumerable<T> collection)
    {
        _headNode = null;
        _tailNode = null;
        _size = 0;
        foreach (var item in collection)
        {
            Enqueue(item);
        }
    }

    IEnumerator<T> IEnumerable<T>.GetEnumerator()
    {
        Node<T>? currentNode = _headNode;
        while (currentNode != null)
        {
            yield return currentNode.Value;
            currentNode = currentNode.Next;
        }
    }

    public IEnumerator GetEnumerator()
    {
        return new MyEnumerator(_headNode);
    }

    public void CopyTo(T[]? array, int index)
    {
        Node<T>? activeNode = _headNode;
        if (array == null)
            throw new ArgumentNullException("Value can not be null");
        else if (index < 0 || index > array.Length - 1)
            throw new ArgumentOutOfRangeException("Index can not be out of
range");
        else if (array.Length - index + 1 < _size)
            throw new ArgumentException("Number of the elements in CustomQueue is
greater than the Array can contain");

        while (activeNode != null)
        {
            array[index] = activeNode.Value;
```

```
            activeNode = activeNode.Next;
            index++;
        }
        OnCopiedTo?.Invoke(this, EventArgs.Empty);
    }

    public void Enqueue(T item)
    {
        Node<T>? node = new Node<T>(item);
        if (_headNode == null)
        {
            _headNode = node;
            _tailNode = node;
        }

        else
        {
            _tailNode.Next = new Node<T>(item);
            _tailNode = _tailNode.Next;
        }

        _size++;
        OnEnqueued?.Invoke(this, new EventArgs());
    }

    public T Dequeue()
    {
        if (_headNode == null)
            throw new InvalidOperationException("CustomQueue is empty");

        T headNodeValue = _headNode.Value;
        _headNode = _headNode.Next;

        _size--;
        OnDequeued?.Invoke(this, EventArgs.Empty);
        return headNodeValue;

    }

    public void Clear()
    {
        _headNode = null;
        _tailNode = null;
        OnCleared?.Invoke(this, EventArgs.Empty);
    }

    public bool Contains(T item)
    {
        Node<T>? activeNode = _headNode;
        while (activeNode != null)
        {
            if (activeNode.Value.Equals(item))
                return true;
            activeNode = activeNode.Next;
        }

        return false;
    }

    public T Peek()
    {
        if (_size == 0)
            throw new InvalidOperationException("CustomQueue is empty");

        return _headNode.Value;
    }

    public T[] ToArray()
    {
```

```
        Node<T>? activeNode = _headNode;
        T[] array = new T[_size];
        for (int i = 0; i < _size; i++)
        {
            array[i] = activeNode.Value;
            activeNode = activeNode.Next;
        }

        return array;
    }

    public CustomQueue<T> Reverse()
    {
        CustomQueue<T> reversedQueue;
        IEnumerable<T> tempArray;
        tempArray = this.ToArray().Reverse();
        reversedQueue = new CustomQueue<T>(tempArray);
        OnReversed?.Invoke(this, EventArgs.Empty);
        return reversedQueue;
    }

    private class MyEnumerator: IEnumerator<T>
    {
        private Node<T>? _activeNode;
        private Node<T>? _headNode;
        private bool _beganEnumerating = false;

        public T Current => _activeNode.Value;

        object IEnumerator.Current => Current;

        public MyEnumerator(Node<T>? headNode)
        {
            _headNode = headNode;
            _activeNode = _headNode;
        }

        public bool MoveNext()
        {
            if (_headNode == null)
                return false;

            else if(!_beganEnumerating)
            {
                _beganEnumerating = true;
                _activeNode = _headNode;
                return true;
            }

            bool hasNext = _activeNode.Next != null;
            if (!hasNext)
            {
                return false;
            }

            _activeNode = _activeNode.Next;
            return hasNext;
        }

        public void Reset()
        {
            _activeNode = _headNode;
        }

        public void Dispose()
        {

        }
```

```
    }
}
```

Файл Program.cs:

```
using System.Numerics;
using MyLibrary;

void EnqueueHandler (object? sender, EventArgs e) => Console.WriteLine("Enquque
operation completed");
void DequeueHandler (object? sender, EventArgs e) => Console.WriteLine("Dequque
operation completed");
void ClearHandler (object? sender, EventArgs e) => Console.WriteLine("Clear
operation completed");
void CopiedToHandler (object? sender, EventArgs e) => Console.WriteLine("Copy
operation completed");
void ReversedHandler (object? sender, EventArgs e) => Console.WriteLine("Reverse
operation completed");

void EnqueueTest<T>(T enqueueElement, IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnEnqueued += EnqueueHandler;
    cqueue.Enqueue(enqueueElement);

    Console.WriteLine("Enqueued elements:");
    foreach(var element in cqueue)
        Console.WriteLine(element);
}
EnqueueTest(12, new int[]{});

void DequeueTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnDequeued += DequeueHandler;
    var poped_element = cqueue.Dequeue();

    Console.WriteLine($"Dequeued element is: {poped_element}");
    Console.WriteLine("Collection elements are:");
    foreach(var element in cqueue)
        Console.WriteLine(element);
}
DequeueTest(new List<int>(){1, 12, 54, 190});

void ClearTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnCleared += ClearHandler;
    cqueue.Clear();

    Console.WriteLine("Collection elements are:");
    foreach(var element in cqueue)
        Console.WriteLine(element);
}
ClearTest(new List<string>(){"a", "b", "c"});

void ContainsTest<T>(T checkedElement, IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    var hasElement = cqueue.Contains(checkedElement);
    Console.WriteLine($"has {checkedElement}: {hasElement}");
}
ContainsTest(5, new []{3, 5, 11});

void ToArrayTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
```

```
    T[] arr = cqueue.ToArray();

    Console.WriteLine("Array elements are:");
    foreach (var el in arr)
        Console.WriteLine(el);
}
ToArrayTest(new bool[]{true, false, false, true});

void ReverseTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnReversed += ReversedHandler;
    cqueue = cqueue.Reverse();

    Console.WriteLine("Collection elements are:");
    foreach (var el in cqueue)
        Console.WriteLine(el);
}
ReverseTest(new int[]{11, 13, 15, 17});

void CopyToTest<T>(T[] insertArray, int startIndex, IEnumerable<T>? list) where T
: IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnCopiedTo += CopiedToHandler;
    cqueue.CopyTo(insertArray, startIndex);

    Console.WriteLine("Array elements are:");
    foreach (var el in insertArray)
        Console.WriteLine(el);
}

CopyToTest(new int[]{10, 11, 12, 13, 14, 15, 16}, 2, new List<int>(){100, 101,
102});

void PeekTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    Console.WriteLine($"Peek element is {cqueue.Peek()}");
}

PeekTest(new List<string>(){"a", "b", "c"});
```

Тестування програми:

Тестування методу Enqueue:

```
void EnqueueTest<T>(T enqueueElement, IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnEnqueued += EnqueueHandler;
    cqueue.Enqueue(enqueueElement);

    Console.WriteLine("Enqueued elements:");
    foreach(var element :object? in cqueue)
        Console.WriteLine(element);
}
EnqueueTest( enqueueElement: 12, list: new int []{});
```

Результат:

```
Enquque operation completed
Enqueued elements:
12
```

Тестування методу Dequeue:

```
void DequeueTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnDequeued += DequeueHandler;
    var poped_element :T = cqueue.Dequeue();

    Console.WriteLine($"Dequeued element is: {poped_element}");
    Console.WriteLine("Collection elements are:");
    foreach(var element :object? in cqueue)
        Console.WriteLine(element);
}
DequeueTest(list: new List<int>(){1, 12, 54, 190});
```

Результат:

```
Dequque operation completed
Dequeued element is: 1
Collection elements are:
12
54
190

Process finished with exit code 0.
```

Тестування методу Clear:

```
void ClearTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnCleared += ClearHandler;
    cqueue.Clear();

    Console.WriteLine("Collection elements are:");
    foreach(var element :object? in cqueue)
        Console.WriteLine(element);
}
ClearTest(list: new List<string>(){"a", "b", "c"});
```

Результат:

```
Clear operation completed
Collection elements are:

Process finished with exit code 0.
```

Тестування методу Contains:

```
void ContainsTest<T>(T checkedElement, IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    var hasElement = cqueue.Contains(checkedElement);
    Console.WriteLine($"has {checkedElement}: {hasElement}");
}

ContainsTest( checkedElement: 5,  list: new int[]{3, 5, 11});
```

Результат:

```
has 5: True

Process finished with exit code 0.
```

Тестування методу Peek:

```
void PeekTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    Console.WriteLine($"Peek element is {cqueue.Peek()}");
}

PeekTest( list: new List<string>(){"a", "b", "c"});
```

Результат:

```
Peek element is a

Process finished with exit code 0.
```

Тестування методу ToArray:

```
void ToArrayTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    T[] arr = cqueue.ToArray();

    Console.WriteLine("Array elements are:");
    foreach (var el in arr)
        Console.WriteLine(el);
}
ToArrayTest(new bool[]{true, false, false, true});
```

Результат:

```
Array elements are:
True
False
False
True

Process finished with exit code 0.
```

Тестування методу Reverse:

```
void ReverseTest<T>(IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnReversed += ReversedHandler;
    cqueue = cqueue.Reverse();

    Console.WriteLine("Collection elements are:");
    foreach (var el in cqueue)
        Console.WriteLine(el);
}
ReverseTest(new int[]{11, 13, 15, 17});
```

Результат:

```
Reverse operation completed
Collection elements are:
17
15
13
11

Process finished with exit code 0.
```

Тестування методу CopyTo:

```
void CopyToTest<T>(T[] insertArray, int startIndex, IEnumerable<T>? list) where T : IComparable
{
    CustomQueue<T> cqueue = new CustomQueue<T>(list);
    cqueue.OnCopiedTo += CopiedToHandler;
    cqueue.CopyTo(insertArray, startIndex);

    Console.WriteLine("Array elements are:");
    foreach (var el:T in insertArray)
        Console.WriteLine(el);
}

CopyToTest(insertArray:new int[]{10, 11, 12, 13, 14, 15, 16}, startIndex:2, list:new List<int>(){100, 101, 102});
```

Результат:

```
Copy operation completed
Array elements are:
10
11
100
101
102
15
16

Process finished with exit code 0.
```

Висновок:

У ході виконання даної лабораторної роботи я запрограмував власну колекцію типу черга(Queue) використовуючи узагальнені типи. Розглянув основні методи даної колекції та імплементував їх у власній колекції. Проаналізував інтерфейси які реалізуються колекціями, та реалізував їх у власній колекції. Створив події та ініціював їх у відповідних методах.

Загалом поглибив знання про колекції та покращив практичні навички їх створення.