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 Lab1;

public class Node<T>
{
    public Node<T> next;
    public T value;

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

Файл CustomQueue.cs:

```
using System.Collections;


namespace Lab1;

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

    private event EventHandler? OnEnqueued = (sender, eventArgs)
        => Console.WriteLine("Enquque operation completed");
    private event EventHandler? OnDequeued = (sender, eventArgs)
        => Console.WriteLine("Dequeue operation completed");
    private event EventHandler? OnCleared = (sender, eventArgs)
        => Console.WriteLine("Clear operation completed");
    private event EventHandler? OnCopiedTo = (sender, eventArgs)
        => Console.WriteLine("Copy operation completed");
    private event EventHandler? OnReversed = (sender, eventArgs)
        => Console.WriteLine("Reverse operation completed");

    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, new EventArgs());
    }

    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, new EventArgs());
        return headNodeValue;

    }

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

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

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()
        {

        }
    }

}
```

Тестування програми:
Ініціалізація екземпляру класу за допомогою конструктора:

```
using Lab1;

CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
```

Результат:

```
Enquque operation completed
Enquque operation completed
Enquque operation completed
Enquque operation completed
```

Тестування методу Enqueue:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
cqueue.Enqueue(item: 5);
foreach(var element :object? in cqueue)
    Console.WriteLine(element);
```

Результат:

```
1
2
3
4
5

Process finished with exit code 0.
```

Тестування методу Dequeue:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
var poped_element :int = cqueue.Dequeue();
Console.WriteLine($"Dequeued element is: {poped_element}");
Console.WriteLine("Collection elements are:");
foreach(var element :object? in cqueue)
    Console.WriteLine(element);
```

Результат:

```
Dequeue operation completed
Dequeued element is: 1
Collection elements are:
2
3
4

Process finished with exit code 0.
```

Тестування методу Clear:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
cqueue.Clear();
foreach(var element :object? in cqueue)
    Console.WriteLine(element);
```

Результат:

```
Clear operation completed

Process finished with exit code 0.
```

Тестування методу Contains:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
var hasThree = cqueue.Contains(item: 3);
var hasTen = cqueue.Contains(item: 10);
Console.WriteLine($"has 3: {hasThree}, has 10: {hasTen}");
```

Результат:

```
has 3: True, has 10: False

Process finished with exit code 0.
```

Тестування методу Peek:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
Console.WriteLine(cqueue.Peek());
```

Результат:

```
1

Process finished with exit code 0.
```

Тестування методу ToArray:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
int[] arr = cqueue.ToArray();
foreach (var el:int in arr)
    Console.WriteLine(el);
```

Результат:

```
1
2
3
4

Process finished with exit code 0.
```

Тестування методу Reverse:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
cqueue = cqueue.Reverse();
foreach (var el:object? in cqueue)
    Console.WriteLine(el);
```

Результат:

```
4
3
2
1

Process finished with exit code 0.
```

Тестування методу CopyTo:

```
CustomQueue<int> cqueue = new CustomQueue<int>(collection: new List<int>(){1, 2, 3, 4});
int[] arr = new int[] {11, 12, 13, 14, 15, 16, 17};
cqueue.CopyTo(arr, index: 1);
foreach (var el: int in arr)
    Console.WriteLine(el);
```

Результат:

```
Copy operation completed
11
1
2
3
4
16
17

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 від яких наслідуються колекції, 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.