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

Звіт
з лабораторної роботи №1 з дисципліни
«Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

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

Варіант 2

Виконав студент ІП-13 Дем'янчук Олександр Петрович
(шифр, прізвище, ім'я, по батькові)

Перевірив Бардін В.
(прізвище, ім'я, по батькові)

Київ 2023

Лабораторна робота №1

Варіант 2

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

Мета: навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

Постановка задачі

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.

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

Виконання завдань

Код програми

```
// MyQueue.cs
using System.Collections;
using System.Diagnostics.CodeAnalysis;
using WebNetLab1.Collections.EventArgs;

namespace WebNetLab1.Collections;
```

```
public class MyQueue<T> : IEnumerable<T>, ICollection
{
    private MyQueueNode? _head;
    private MyQueueNode? _tail;

    public event EventHandler<PeekEventArgs<T>> PeekEvent;
    public event EventHandler<QueueEmptyEventArgs> QueueEmptyEvent;

    public MyQueue()
    {
        _head = null;
        _tail = null;
    }

    public MyQueue(IEnumerable<T> source)
    {
        if (source is null)
        {
            throw new ArgumentNullException(nameof(source));
        }
        foreach (var item in source)
        {
            Enqueue(item);
        }
    }

    public int Count
    {
        get
        {
            int count = 0;
            var current = _head;
```

```
            while (current is not null)
            {
                count++;
                current = current.Next;
            }

            return count;
        }
    }

    public bool IsSynchronized => false;
    public object SyncRoot => this;

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

        if (arrayIndex < 0 || arrayIndex > array.Length)
        {
            throw new ArgumentOutOfRangeException(nameof(arrayIndex),
arrayIndex,
                "Index is out of bounds of this array.");
        }

        if (array.Length - arrayIndex < Count)
        {
            throw new ArgumentException("There's not enough space to copy
into this range of an array.");
        }
```

```
    if (Count == 0)
    {
        return;
    }

    var current = _head;
    int index = arrayIndex;
    while (current is not null)
    {
        array[index] = current.Data;
        index++;
        current = current.Next;
    }
}

public void Clear()
{
    _head = null;
    _tail = null;
    OnQueueEmpty(new QueueEmptyEventArgs("A queue was cleared."));
}

void ICollection.CopyTo(Array array, int arrayIndex)
{
    if (array is null)
    {
        throw new ArgumentNullException(nameof(array));
    }

    if (array.Rank != 1)
    {
```

```
            throw new ArgumentException("The array has an invalid rank.");
        }

        if (array.GetLowerBound(0) != 0)
        {
            throw new ArgumentException("The array must have a lower bound
of 0.");
        }

        if (arrayIndex < 0 || arrayIndex > array.Length)
        {
            throw new ArgumentOutOfRangeException(nameof(arrayIndex),
arrayIndex,
                "Index is out of bounds of this array.");
        }

        if (array.Length - arrayIndex < Count)
        {
            throw new ArgumentException("There's not enough space to copy
into this range of an array.");
        }

        if (Count == 0)
        {
            return;
        }

        var current = _head;
        int index = arrayIndex;
        while (current is not null)
        {
            array.SetValue(current.Data, index);
```

```
            index++;
            current = current.Next;
        }
    }

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

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

    public void Enqueue(T item)
    {
        var newNode = new MyQueueNode(item);
        if (_head is null)
        {
            _head = newNode;
            _tail = _head;
            return;
        }

        _tail.Next = newNode;
        _tail = newNode;
```

```
}

public T Dequeue()
{
    if (_head is null)
    {
        throw new InvalidOperationException("The queue is empty.");
    }

    var removedData = HandleDequeue();

    return removedData;
}

public bool TryDequeue([MaybeNullWhen(false)] out T result)
{
    if (_head is null)
    {
        result = default;
        return false;
    }

    result = HandleDequeue();
    return true;
}

public T Peek()
{
    if (_head is null)
    {
        throw new InvalidOperationException("The queue is empty.");
    }
```

```
            var result = _head.Data;
            OnPeek(new PeekEventArgs<T>("An element was retrieved from Peek method.", result));
            return result;
        }

        public bool TryPeek([MaybeNullWhen(false)] out T result)
        {
            if (_head is null)
            {
                result = default;
                return false;
            }

            result = _head.Data;
            OnPeek(new PeekEventArgs<T>("An element was retrieved from TryPeek method.", result));
            return true;
        }

        public bool Contains(T item)
        {
            var current = _head;
            while (current is not null)
            {
                if (current.Data.Equals(item))
                {
                    return true;
                }

                current = current.Next;
```

```
            }

            return false;
        }

        public T[] ToArray()
        {
            if (Count == 0)
            {
                return Array.Empty<T>();
            }

            var array = new T[Count];

            CopyTo(array, 0);

            return array;
        }

        private T HandleDequeue()
        {
            var dequeuedData = _head.Data;

            if (_head == _tail)
            {
                _head = null;
                _tail = null;

                OnQueueEmpty(new QueueEmptyEventArgs("The last element was
dequeued."));
            }
            else
```

```
        {
            _head = _head.Next;
        }

        return dequeuedData;
    }

    private void OnQueueEmpty(QueueEmptyEventArgs e)
    {
        QueueEmptyEvent?.Invoke(this, e);
    }
    private void OnPeek(PeekEventArgs<T> e)
    {
        PeekEvent?.Invoke(this, e);
    }
    private class MyQueueNode
    {
        public T Data { get; }
        public MyQueueNode? Next { get; internal set; }

        public MyQueueNode(T data)
        {
            Data = data;
            Next = null;
        }
    }
}

// PeekEventArgs.cs
namespace WebNetLab1.Collections.EventArgs;

public class PeekEventArgs<T> : System.EventArgs
```

```
{
    public string Message { get; }
    public T Data { get; }

    public PeekEventArgs(string message, T data)
    {
        Message = message;
        Data = data;
    }

}

// QueueEmptyEventArgs.cs
namespace WebNetLab1.Collections.EventArgs;

public class QueueEmptyEventArgs : System.EventArgs
{
    public string Message { get; }

    public QueueEmptyEventArgs(string message)
    {
        Message = message;
    }

}

// Program.cs
using WebNetLab1.Collections;
using WebNetLab1.Collections.EventArgs;

public class Program
{
```

```
static void Main()
{
    void DisplayCollection(IEnumerable<int> collection)
    {
        foreach (var item in collection)
        {
            Console.Write($"{item} ");
        }

        Console.WriteLine();
    }

    void OnPeek(object? sender, PeekEventArgs<int> e)
    {
        Console.WriteLine($"Peek event: {e.Message}, data: {e.Data}");
    }

    void OnQueueEmpty(object? sender, QueueEmptyEventArgs e)
    {
        Console.WriteLine($"Queue empty event: {e.Message}");
    }

    var queue = new MyQueue<int>();
    queue.PeekEvent += OnPeek;
    queue.QueueEmptyEvent += OnQueueEmpty;

    for (int i = 0; i < 10; i++)
    {
        queue.Enqueue(i);
    }
    Console.WriteLine("Enqueue check, queue:");
    DisplayCollection(queue);
```

```
        Console.WriteLine("Peek check, front element: {0}", queue.Peek());

        if (queue.TryPeek(out var result))
        {
            Console.WriteLine("TryPeek check, front element: {0}", result);
        }

        Console.WriteLine("Contains check before dequeue, does the queue
contain 3? {0}", queue.Contains(3));

        Console.WriteLine("Dequeue check, dequeued elements:");
        for (int i = 0; i < 5; i++)
        {
            Console.WriteLine(queue.Dequeue());
        }

        Console.WriteLine("Contains check after dequeue, does the queue
contain 3? {0}", queue.Contains(3));

        var array = queue.ToArray();
        Console.WriteLine("ToArray check, array:");
        DisplayCollection(array);

        Console.WriteLine("TryDequeue check, dequeued elements:");
        for (int i = 0; i < 3; i++)
        {
            queue.TryDequeue(out var item);
            Console.WriteLine(item);
        }

        Console.WriteLine("After TryDequeue check, queue:");
```

```
DisplayCollection(queue);

Console.WriteLine("TryDequeue event trigger, last elements:");
for (int i = 0; i < 2; i++)
{
   queue.TryDequeue(out var item);
   Console.WriteLine(item);
}

Console.WriteLine("Trying to initialize a queue from null source:");
try
{
   var queueFromNull = new MyQueue<int>(null);
}
catch (ArgumentNullException e)
{
   Console.WriteLine(e.Message);
}

var queueFromArray = new MyQueue<int>(new int[] { 1, 3, 5, 7, 9 });
Console.WriteLine("Trying to initialize a queue from array, queue:");
DisplayCollection(queueFromArray);


var shortArray = new int[4];
Console.WriteLine("CopyTo check on short array:");
try
{
   queueFromArray.CopyTo(shortArray, 0);
}
catch (ArgumentException e)
{
```

```
            Console.WriteLine(e.Message);
        }

        var longArray = new int[5];
        queueFromArray.CopyTo(longArray, 0);
        Console.WriteLine("CopyTo check on long array, array:");
        DisplayCollection(longArray);

        queueFromArray.QueueEmptyEvent += OnQueueEmpty;
        queueFromArray.Clear();
        Console.WriteLine("Clear check, queue:");
        DisplayCollection(queueFromArray);
    }
}
```

Результат виконання коду Program.cs

```
Enqueue check, queue:
0 1 2 3 4 5 6 7 8 9
Peek event: An element was retrieved from Peek method., data: 0
Peek check, front element: 0
Peek event: An element was retrieved from TryPeek method., data: 0
TryPeek check, front element: 0
Contains check before dequeue, does the queue contain 3? True
Dequeue check, dequeued elements:
0
1
2
3
4
Contains check after dequeue, does the queue contain 3? False
ToArray check, array:
5 6 7 8 9
TryDequeue check, dequeued elements:
5
6
7
After TryDequeue check, queue:
8 9
TryDequeue event trigger, last elements:
8
Queue empty event: The last element was dequeued.
9
Trying to initialize a queue from null source:
Value cannot be null. (Parameter 'source')
Trying to initialize a queue from array, queue:
1 3 5 7 9
CopyTo check on short array:
There's not enough space to copy into this range of an array.
CopyTo check on long array, array:
1 3 5 7 9
Queue empty event: A queue was cleared.
```

Висновки:

Висновок: В рамках лабораторної роботи №1 було створено узагальнену колекцію Черга зі збереженням даних у вигляді динамічного зв’язного списку, та реалізацією інтерфейсів неймспейсу System.Collections. Було реалізовано

необхідні методи, оброблено виключні ситуації та реалізовано роботу подій в колекції. Функціонал колекції було продемонстровано в коді файлу Program.cs.