Міністерство освіти і науки України КПІ ім. Ігоря Сікорського

Факультет Інформатики та Обчислювальної Техніки

ЗВІТ

до лабораторної роботи №1 з модуля

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

Перевірив:
Викладач кафедри ІСТ
ФІОТ
Бардін В.

Виконав:
Боровков Іван
гр. ІП-11

2023

Узагальнені типи (Generic) з підтримкою подій. Колекції

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.

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

Код програми

IQueue.cs

```
namespace MyQueue.Interfaces;

public interface IQueue<T>
{

    public void Clear();

    public bool Contains(T item);

    public void Enqueue(T item);

    public T Dequeue();

    public T Peek();
```

```
    public T[] ToArray();
}
```

MyQueue.cs

```
using System.Collections;
using System.ComponentModel;
using MyQueue.Interfaces;

namespace MyQueue;

public class MyQueue<T> : IEnumerable<T>, ICollection, IQueue<T>
{
    private Node? _head;
    private Node? _tail;
    private int _size = 0;

    public event EventHandler<CollectionChangeEventArgs>? ItemAdded;
    public event EventHandler<CollectionChangeEventArgs>? ItemDeleted;
    public event EventHandler<CollectionChangeEventArgs>? QueueCleared;


    public int Count => _size;
    public bool IsSynchronized => false;
    public object SyncRoot => this;

    public void Enqueue(T item)
    {
        if (_head is null)
        {
            _head = new Node()
            {
                Value = item,
                Next = null
            };
            _tail = _head;
        }
        else
        {
```

```
            var newNode = new Node()
            {
                Value = item,
                Next = null
            };
            _tail!.Next = newNode;          //Will never be null if head isn't null
            _tail = _tail.Next;
        }
        _size++;

        ItemAdded?.Invoke(this, new
CollectionChangeEventArgs(CollectionChangeAction.Add, item));
    }



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

        var removed = _head.Value;
        _head = _head.Next;

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

        _size--;

        ItemDeleted?.Invoke(this, new
CollectionChangeEventArgs(CollectionChangeAction.Remove, removed));

        return removed;
    }

    public T Peek()
```

```
    {
        if (_head is null)
        {
            throw new InvalidOperationException("Queue is empty.");
        }

        return _head.Value;
    }

    public void Clear()
    {
        _size = 0;
        _head = null;
        _tail = null;
        QueueCleared?.Invoke(this, new
CollectionChangeEventArgs(CollectionChangeAction.Refresh, null));
    }

    public bool Contains(T item)
    {
        if (_head is null) return false;

        var currentNode = new Node
        {
            Value = _head.Value,
            Next = _head.Next
        };

        while (currentNode is not null)
        {
            //Why is this warning here????
            if (currentNode.Value!.Equals(item))
            {
                return true;
            }

            currentNode = currentNode.Next;
        }

        return false;
```

```
    }

    public void CopyTo(Array array, int index)
    {
        ArgumentNullException.ThrowIfNull(array);

        if (index < 0)
        {
            throw new ArgumentOutOfRangeException(nameof(index), index, "Index
must be greater than 0.");
        }

        if (index > array.Length)
        {
            throw new ArgumentOutOfRangeException(nameof(index), index,
                "Index must be less or equal than size of array.");
        }

        if (array.Length - index < _size)
        {
            throw new ArgumentException("Invalid length of the array.");
        }

        if (_head is null)
        {
            return;
        }

        var sourceArray = ToArray();

        Array.Copy(sourceArray, 0, array, index, _size);
    }

    public T[] ToArray()
    {
        if (_head is null)
        {
            return Array.Empty<T>();
        }
```

```
    var currentNode = new Node()
    {
        Value = _head.Value,
        Next = _head.Next
    };

    var array = new T[_size];
    int i = 0;

    while (currentNode is not null)
    {
        array[i++] = currentNode.Value;
        currentNode = currentNode.Next;
    }

    return array;
}

public IEnumerator<T> GetEnumerator()
{
    return new MyEnumerator(this);
}

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

private class MyEnumerator : IEnumerator<T>
{
    private readonly MyQueue<T> _queue;
    private Node? _currentNode;
    private bool _ended;

    internal MyEnumerator(MyQueue<T> queue)
    {
        _queue = queue;
        _ended = false;
        _currentNode = null;
    }
```

```
public T Current
{
    get
    {
        if (_currentNode is null)
        {
            throw new InvalidOperationException("Enum wasn't started");
        }
        if (_ended)
        {
            throw new InvalidOperationException("Enum has already ended");
        }

        return _currentNode.Value;
    }
}

object IEnumerator.Current => Current!;

public bool MoveNext()
{
    if (_ended)
    {
        return false;
    }

    if (_currentNode is null)
    {
        if (_queue._head is null)
        {
            return false;
        }

        _currentNode = _queue._head;
        return true;
    }

    if (_currentNode.Next is null)
    {
```

```
            _ended = true;
            return false;
        }

        _currentNode = _currentNode.Next;
        return true;
    }

    public void Reset()
    {
        _ended = false;
        _currentNode = null;
    }

    public void Dispose()
    {
        _ended = true;
    }
}

private class Node
{
    public required T Value { get; set; }
    public required Node? Next { get; set; }
}

}
```

Висновок: в ході роботи ми навчилися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.