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”

Виконав(ла): Маланічев Д., ІС-13

Перевірив(ла): Бардін В.

Київ 2023

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.

Хід роботи

Варіант:

9	Динамічний масив з довільним діапазоном індексу	Див. List<T>	Збереження даних за допомогою вектору

Лістинг коду:

DynamicArray.cs

```
using System.Collections;
using MyCollection.EventsArgs;

namespace MyCollection;

public class DynamicArray<T> : IList<T>, IReadOnlyList<T>
{
    private const int DefaultCapacity = 16;

    public int Count => _size;
    public bool IsReadOnly => false;

    private int _size;
    private int _capacity;
    private T[] _items;

    public event EventHandler<ItemManipulationEventArgs<T>> ItemAdded =
null!;
    public event EventHandler<ItemManipulationEventArgs<T>> ItemRemoved =
null!;
    public event EventHandler<DynamicArrayResizedEventArgs>
DynamicArrayResized = null!;

    protected virtual void OnItemAdded(T item, int index)
    {
        // ReSharper disable once
ConditionalAccessQualifierIsNonNullableAccordingToAPIContract
        ItemAdded?.Invoke(this, new ItemManipulationEventArgs<T>(item,
index));
    }

    protected virtual void OnItemRemoved(T item, int index)
    {
        // ReSharper disable once
ConditionalAccessQualifierIsNonNullableAccordingToAPIContract
        ItemRemoved?.Invoke(this, new ItemManipulationEventArgs<T>(item,
index));
    }

    protected virtual void OnDynamicArrayResized(int oldSize, int newSize)
    {
        // ReSharper disable once
ConditionalAccessQualifierIsNonNullableAccordingToAPIContract
        DynamicArrayResized?.Invoke(this, new
DynamicArrayResizedEventArgs(oldSize, newSize));
    }
```

```
public T this[int index]
{
    get => _items[index];
    set
    {
        if(index >= _size)
        {
            throw new ArgumentException("Invalid index");
        }

        _items[index] = value;
    }
}

public DynamicArray(int capacity = DefaultCapacity)
{
    if (capacity < 0)
    {
        throw new ArgumentOutOfRangeException(nameof(capacity));
    }

    _capacity = capacity;
    _size = 0;
    _items = capacity is 0
        ? Array.Empty<T>()
        : new T[capacity];
}

public DynamicArray(IEnumerable<T> items)
{
    if (items is null)
    {
        throw new ArgumentNullException(nameof(items));
    }

    // Convert items to List<T> to suppress possible multiple enumeration
    var list = items.ToList();

    // Setting _capacity to the size of items to prevent needless resizes
    _capacity = list.Count;

    _items = new T[_capacity];
    _size = 0;

    foreach (var item in list)
    {
        Add(item);
    }
}

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

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

public void Add(T item)
{
    if (_size >= _capacity)
```

```
        {
            Resize();
        }

        _items[_size] = item;
        _size++;

        OnItemAdded(item, _size);
    }

    public void AddRange(IEnumerable<T> items)
    {
        if (items is null)
        {
            throw new ArgumentNullException(nameof(items));
        }

        foreach (var item in items)
        {
            Add(item);
        }
    }

    public void Clear()
    {
        _items = new T[DefaultCapacity];
        _capacity = _size = 0;
    }

    public bool Contains(T item)
    {
        for (int i = 0; i < _size; i++)
        {
            var element = _items[i];
            if (element?.Equals(item) == true)
            {
                return true;
            }
        }

        return false;
    }

    public void CopyTo(T[] array, int arrayIndex)
    {
        if (array.Length - arrayIndex < _items.Length)
        {
            throw new ArgumentException("Dest array is too small");
        }

        Array.Copy(_items, array, _items.Length);
    }

    public bool Remove(T item)
    {
        var index = Array.IndexOf(_items, item);
        var isRemoved = index != -1;

        RemoveAt(index);

        return isRemoved;
    }

    public int IndexOf(T item)
```

```
    {
        return Array.IndexOf(_items, item);
    }

    public void Insert(int index, T item)
    {
        if (_size < index)
        {
            throw new InvalidOperationException("Invalid index");
        }

        if (_size == _capacity)
        {
            Resize();
        }

        if (_size == index)
        {
            _items[index] = item;
        }

        Array.Copy(_items, index, _items, index + 1, _size - index);
        _size++;
        _items[index] = item;
    }

    public void RemoveAt(int index)
    {
        if (index < 0 || index > _size)
        {
            throw new ArgumentOutOfRangeException(nameof(index));
        }

        var item = _items[index];

        _size--;
        Array.Copy(_items, index + 1, _items, index, _size - index);

        OnItemRemoved(item, index);
    }

    private void Resize()
    {
        var newCapacity = _capacity * 2;
        var tempArray = new T[newCapacity];
        Array.Copy(_items, tempArray, _size);
        _items = tempArray;
        _capacity = newCapacity;

        OnDynamicArrayResized(newCapacity / 2, newCapacity);
    }
}
```

DynamicArrayEnumerator.cs

```
using System.Collections;

namespace MyCollection;

internal class DynamicArrayEnumerator<T> : IEnumerator<T>
{
    private readonly IList<T> _list;
    private int _cursor;
    private T _current;
    public T Current => _current;
    object IEnumerator.Current => _current!;

    public DynamicArrayEnumerator(IList<T> list)
    {
        _list = list;
        _cursor = 0;
        _current = _list.Any() ? _list[_cursor] : default!;
    }

    public bool MoveNext()
    {
        if (_cursor < _list.Count)
        {
            _current = _list[_cursor];
            _cursor++;
            return true;
        }

        return false;
    }

    public void Reset()
    {
        _cursor = 0;
        _current = _list[0];
    }




    public void Dispose()
    {
    }
}
```

DynamicArrayResizedEventArgs.cs

```
namespace MyCollection.EventsArgs;

public class DynamicArrayResizedEventArgs : EventArgs
{
    public int OldCapacity { get; private set; }
    public int NewCapacity { get; private set; }

    public DynamicArrayResizedEventArgs(int oldCapacity, int newCapacity)
    {
        OldCapacity = oldCapacity;
        NewCapacity = newCapacity;
    }
}
```

ItemManipulationeventArgs.cs

```
namespace MyCollection.EventsArgs;

public class ItemManipulationEventArgs<T> : EventArgs
{
    public T Item { get; private set; }
    public int Index { get; private set; }

    public ItemManipulationEventArgs(T item, int index)
    {
        Item = item;
        Index = index;
    }
}
```

Program.cs

```
namespace MyCollection
{
    public static class Program
    {
        private static void Main()
        {
            var list = new DynamicArray<int>(5) { 1, 2, 3, 4, 5 };

            Console.WriteLine("Iterate using foreach loop:");
            PrintList(list);

            Console.WriteLine("Indexers:");
            Console.Write(list[0] + " ");
            Console.Write(list[1] + " ");
            Console.Write(list[2] + " ");
            Console.Write(list[3] + " ");
            Console.Write(list[4] + " ");
            Console.WriteLine();

            Console.WriteLine("Remove at index 1:");
            list.RemoveAt(1);
            PrintList(list);

            Console.WriteLine("Remove element 3");
            list.Remove(3);
            PrintList(list);

            Console.WriteLine("Add 6, 7, 8");
            list.Add(6);
            list.Add(7);
            list.Add(8);
            PrintList(list);

            Console.WriteLine("Insert 9 to the index of 2");
            list.Insert(2, 9);
            PrintList(list);

            Console.WriteLine("Check if 9 contains in list");
            Console.WriteLine(list.Contains(9) ? "Yes" : "No");

            Console.WriteLine("Find an index of 9");
            Console.WriteLine(list.IndexOf(9));

            var list2 = new DynamicArray<int>(new List<int> { 1, 2, 3, 4, 5,
6 });
            PrintList(list2);

            list.ItemAdded += (sender, args) =>
            {
                Console.WriteLine(
                    $@"New item with value {args.Item} was added at index
{args.Index} (object: {sender}).");
            };

            list.ItemRemoved += (sender, args) =>
            {
                Console.WriteLine(
                    $@"Item with value {args.Item} was removed from index
{args.Index} (object: {sender}).");
            };
```

```
            list.DynamicArrayResized += (sender, args) =>
            {
                Console.WriteLine(
                    $@"Object: {sender} has just resized from capacity
{args.OldCapacity} to {args.NewCapacity} ");
            };

            list.AddRange(new[] { 100, 101, 102, 103, 104, 105 });

            list.Remove(102);
        }

        private static void PrintList(DynamicArray<int> list)
        {
            foreach (var i in list)
            {
                Console.Write(i + " ");
            }

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

Висновок: в ході виконання даної лабораторної роботи було створено власну реалізацію узагальненої колекції, а саме списку. При цьому було використано інтерфейси IList<T> та IList, а також було продемонстровано використання подій (event) та обробку виключних ситуацій. Результат роботи продемонстровано в консольному застосунку, який використовує всі методи створеної колекції.

Посилання на GitHub: https://github.com/DenysMalanichev/DynamicArray-.NET-Lab1-