Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №1

Сучасні технології Web-застосувань на платформі Microsoft.NET

Узагальнені типи (Generic) з підтримкою подій. Колекції

Варіант 9

Виконала:

студентка групи ІА-11
Шурек Ю.К.

Перевірив:

Бардін В.

Київ 2023

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

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

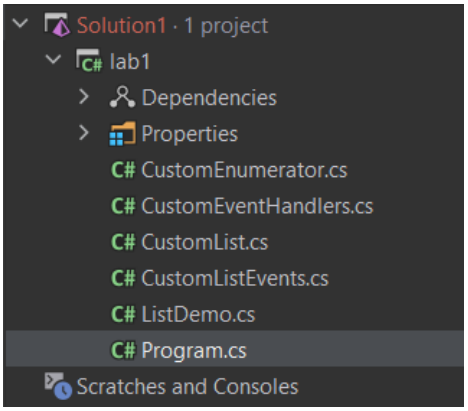
Завдання:

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 типі зберегти у динамічній бібліотеці.
4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

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

Хід роботи:

Структура проекту:



Program.cs

```
using System;

namespace lab1;

internal static class Program
```

```
{
    public static void Main(string[] args)
    {
        try
        {
            ListDemo.ListShowAll();
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }
}
```

ListDemo.cs

```
using System;
using System.Linq;

namespace lab1;

public static class ListDemo
{
    public static void ListShowAll()
    {
        CustomList<int> list = new CustomList<int>();

        list.ItemAdded += CustomEventHandlers.PrintListItemEventHandler!;
        list.ItemRemoved += CustomEventHandlers.PrintListItemEventHandler!;
        list.ListCleared += CustomEventHandlers.PrintListEventHandler!;
        list.ListResized += CustomEventHandlers.PrintListResizedEventHandler!;

        list.Add( 2);
        list.Add(3);
        list.Insert(2, 5);
        list.Insert(2, 4);
        list.Insert(0, 1);

        Console.Write("After adding and inserting elements: ");
        Printing(list);

        Console.WriteLine($"Contains element 3: {list.Contains(3)}");

        Console.WriteLine($"Index of element 5: {list.IndexOf(5)}");

        int[] arrayToCopyTo = new int[list.Count];
        list.CopyTo(arrayToCopyTo, 0);
        Console.Write("Array copied from list: ");
        arrayToCopyTo.ToList().ForEach(item => Console.Write($"{item} "));
        Console.WriteLine();

        list[1] = 10;
        Console.Write("After list[1] = 10: ");
        Printing(list);

        list.Remove(3);
```

```
        list.RemoveAt(2);
        Console.Write("After removing el=3 and at index=2: ");
        Printing(list);

        list.Clear();
        Console.WriteLine($"After clear. Count: {list.Count}");
    }

    private static void Printing<T>(CustomList<T> list)
    {
        string elements = string.Join(", ", list);
        Console.ForegroundColor = ConsoleColor.Magenta;
        Console.WriteLine(elements);
        Console.ForegroundColor = ConsoleColor.Gray;
    }
}
```

CustomList.cs

```
using System;
using System.Collections.Generic;

namespace lab1;

using System.Collections;

public class CustomList<T> : IList<T>
{
    public int Count  => _size;
    public bool IsReadOnly => false;

    #region private fields

    private int _size;
    private T[] _items;
    private int _capacity;
    private const int DefaultCapacity = 4;

    #endregion

    #region implementation of IList<T>
    public CustomList(int capacity = 0)
    {
        if (capacity < 0)
        {
            throw new ArgumentOutOfRangeException(nameof(capacity), "Capacity
cannot be a negative value.");
        }
        _size = 0;
        _capacity = capacity;
        _items = capacity is 0 ? Array.Empty<T>() : new T[capacity];
    }

    public IEnumerator<T> GetEnumerator()
    {
        return new CustomEnumerator<T>(this);
```

```
    }

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

    public void Add(T item)
    {
        if (_size >= _capacity)
        {
            Resize();
        }
        _items[_size] = item;
        _size++;
        OnItemAdded(item, _size - 1);
    }

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

    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 is null)
        {
            throw new ArgumentNullException(nameof(array), "Array cannot be
null.");
        }

        if (arrayIndex < 0 || arrayIndex >= array.Length)
        {
            throw new ArgumentOutOfRangeException(nameof(arrayIndex), "Invalid
array index");
        }

        if (array.Length - arrayIndex < _size)
        {
```

```
            throw new ArgumentException("Number of elements to copy cannot be
placed into the destination array.");
        }

        Array.ConstrainedCopy(_items, 0, array, arrayIndex, _size);
    }

    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 (index < 0 || index > _size)
        {
            throw new ArgumentOutOfRangeException(nameof(index), "Index is out of
range. It must be within the current list size.");
        }
        if (_size == _capacity)
        {
            Resize();
        }
        Array.Copy(_items, index, _items, index + 1, _size - index);
        _items[index] = item;
        _size++;
        OnItemAdded(item, index);
    }

    public void RemoveAt(int index)
    {
        CheckIndex(index);
        Array.Copy(_items, index + 1, _items, index, _size - index - 1);
        _size--;
        OnItemRemoved(this[index], index);
    }

    public T this[int index]
    {
        get => _items[index];
        set
        {
            CheckIndex(index);
            _items[index] = value;
        }
    }

    #endregion
```

```
    #region work with events

    public EventHandler<CustomListItemEventArgs<T>> ItemAdded;

    public EventHandler<CustomListItemEventArgs<T>> ItemRemoved;

    public EventHandler<CustomListBaseEventArgs> ListCleared;

    public EventHandler<CustomListEventArgs> ListResized;

    private void OnItemAdded(T item, int index)
    {
        if (ItemAdded != null)
        {
            ItemAdded(this, new CustomListItemEventArgs<T>(item, index,
ModificationTypes.ItemAdded));
        }
    }

    private void OnItemRemoved(T item, int index)
    {
        if (ItemRemoved != null)
        {
            ItemRemoved(this, new CustomListItemEventArgs<T>(item, index,
ModificationTypes.ItemRemoved));
        }
    }

    private void OnListCleared()
    {
        if (ListCleared != null)
        {
            ListCleared(this, new
CustomListBaseEventArgs(ModificationTypes.ListCleared));
        }
    }

    private void OnListResized(int oldCapacity)
    {
        if (ListResized != null)
        {
            ListResized(this, new CustomListEventArgs(oldCapacity, _capacity));
        }
    }
    #endregion

    #region private methods
    private void Resize()
    {
        var oldCapacity = _capacity;
        var newCapacity = _capacity <= 0 ? DefaultCapacity : _capacity * 2;
        var tempArray = new T [newCapacity];
        Array.Copy(_items, tempArray, _size);
        _items = tempArray;
        _capacity = newCapacity;
```

```
        OnListResized(oldCapacity);
    }
    private void CheckIndex(int index)
    {
        if (index < 0 || index >= _size)
        {
            throw new ArgumentOutOfRangeException(nameof(index), "Index is out of
range. It must be within the current list size.");
        }
    }

    #endregion
}
```

CustomEnumerator.cs

```
using System.Collections;
using System.Collections.Generic;

namespace lab1;

public class CustomEnumerator<T> : IEnumerator<T>
{

    public T Current => _current;
    object IEnumerator.Current => _current!;

    private readonly IList<T> _list;
    private int _cursor;
    private T _current;

    public CustomEnumerator(IList<T> list)
    {
        _list = list;
        _cursor = -1;
        _current = default!;
    }

    public bool MoveNext()
    {
        if (_cursor >= _list.Count - 1)
        {
            return false;
        }

        _cursor++;
        _current = _list[_cursor];
        return true;
    }

    public void Reset()
    {
        _cursor = -1;
        _current = default!;
    }
```

```
    public void Dispose()
    {
    }
}
```

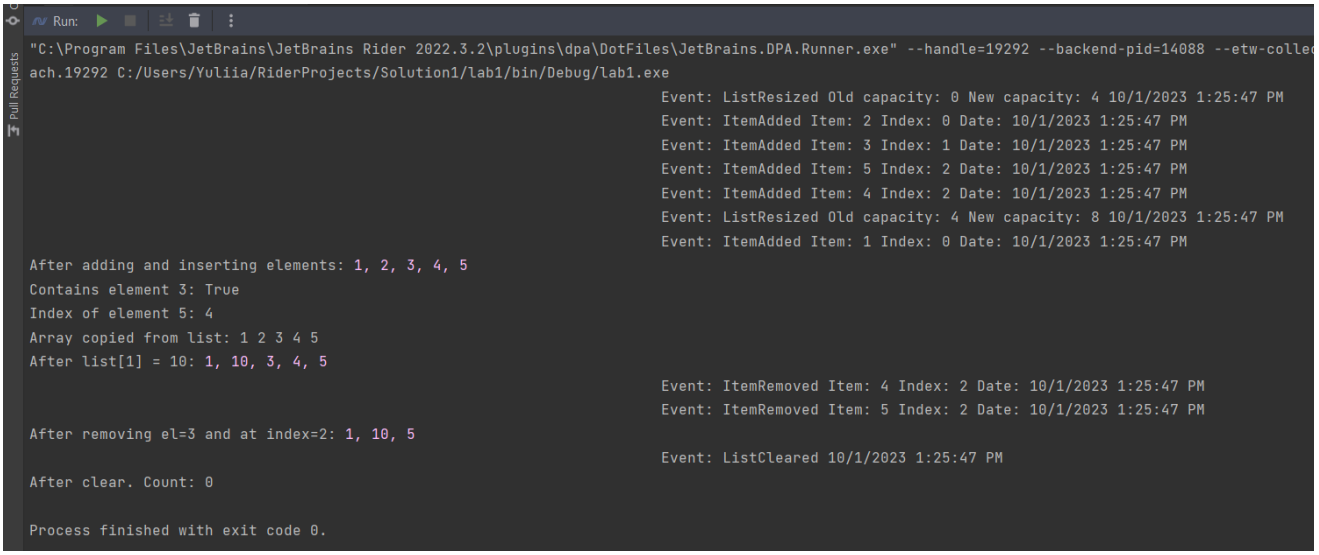
CustomListEvents.cs

```
using System;

namespace lab1;

public enum ModificationTypes
{
    ItemAdded,
    ItemRemoved,
    ListCleared,
    ListResized
}

public class CustomListBaseEventArgs : EventArgs
{
    public ModificationTypes ModificationTypes { get; }
    public DateTime DateTime { get; }

    public CustomListBaseEventArgs(ModificationTypes modificationTypes)
    {
        ModificationTypes = modificationTypes;
        DateTime = DateTime.Now;
    }
}

public class CustomListItemEventArgs<T> : CustomListBaseEventArgs
{
    public T Item { get; }
    public int Index { get; }

    public CustomListItemEventArgs(T item, int index, ModificationTypes
modificationTypes) : base(modificationTypes)
    {
        Item = item;
        Index = index;
    }
}

public class CustomListEventArgs : CustomListBaseEventArgs
{
    public int OldCapacity { get; }
    public int NewCapacity { get; }

    public CustomListEventArgs(int oldCapacity, int newCapacity) :
base(ModificationTypes.ListResized)
    {
        OldCapacity = oldCapacity;
        NewCapacity = newCapacity;
    }
}
```

CustomEventHandlers.cs

```
using System;

namespace lab1;

public static class CustomEventHandlers
{
   public static void PrintListItemEventHandler<T>(object sender,
CustomListItemEventArgs<T> e)
    {
        Console.WriteLine($"\t\t\t\t\t\t\t\t\tEvent: {e.ModificationTypes} Item:
{e.Item} Index: {e.Index} Date: {e.DateTime}");
    }

   public static void PrintListEventHandler(object sender, CustomListBaseEventArgs
e)
    {
        Console.WriteLine($"\t\t\t\t\t\t\t\t\tEvent: {e.ModificationTypes}
{e.DateTime}");
    }

   public static void PrintListResizedEventHandler(object sender,
CustomListEventArgs e)
    {
        Console.WriteLine($"\t\t\t\t\t\t\t\t\tEvent: {e.ModificationTypes} Old
capacity: {e.OldCapacity} New capacity: {e.NewCapacity} {e.DateTime}");
    }
}
```

Результат роботи програми:



Висновок: я навчилася проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.