Національний технічний університет України «КПІ»
Факультет інформатики та обчислювальної техніки
Кафедра Інформаційних систем та технологій

Лабораторна робота №1

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

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

Виконала:
студентка гр. ІО-15
Григор’єв К. С.
Викладач:
Бардін В.

2023 рік

Мета: 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.

Варіант:

5	Динамічний массив	Див. List<T>	Збереження даних за допомогою динамічно зв'язаного списку

Посилання на код GitHub: https://github.com/Grugoriev180/DotNetLabs

Код бібліотеки:

MyArray.cs

```
using DotNetLab1.MyCollections;
using DotNetLab1.MyCollections.CustomEventArgs;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Xml.Linq;

public class DynamicArray<T> : IList<T>
{

	private LinkedListNode head;
	private int count;

	private class LinkedListNode
```

```
	{
		public T Data { get; set; }
		public LinkedListNode Next { get; set; }

		public LinkedListNode(T data)
		{
			Data = data;
			Next = null;
		}
	}
	public DynamicArray()
	{
		head = null;
		count = 0;
	}

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

	public T this[int index]
	{
		get
		{
			if (index < 0 || index >= count)
			{
				throw new IndexOutOfRangeException("Index is out of
range.");
			}
			LinkedListNode current = head;
			for (int i = 0; i < index; i++)
			{
				current = current.Next;
			}
			return current.Data;
		}
		set
		{
			if (index < 0 || index >= count)
			{
				throw new IndexOutOfRangeException("Index is out of
range.");
			}
			LinkedListNode current = head;
			for (int i = 0; i < index; i++)
			{
				current = current.Next;
			}
			current.Data = value;
		}
	}
	#region Events
	public EventHandler<ArrayItemEventArgs<T>> ItemAdded;

	public EventHandler<ArrayItemEventArgs<T>> ItemRemoved;

	public EventHandler<ArrayEventArgs> ArrayCleared;


	protected virtual void OnItemAdded(T item, int index)
	{
		if (ItemAdded != null)
		{
			ItemAdded(this, new ArrayItemEventArgs<T>(item, index,
ArrayAction.Add));
		}
```

```
	}

	protected virtual void OnItemRemoved(T item, int index)
	{
		if (ItemRemoved != null)
		{
			ItemRemoved(this, new ArrayItemEventArgs<T>(item, index,
ArrayAction.Remove));
		}
	}

	protected virtual void OnArrayCleared()
	{
		if (ArrayCleared != null)
		{
			ArrayCleared(this, new ArrayEventArgs(ArrayAction.Clear));
		}
	}

	#endregion
	public void Add(T item)
	{
		LinkedListNode newNode = new LinkedListNode(item);
		if (head == null)
		{
			head = newNode;
		}
		else
		{
			LinkedListNode current = head;
			while (current.Next != null)
			{
				current = current.Next;
			}
			current.Next = newNode;
		}
		count++;
		OnItemAdded(item, Count - 1);
	}



	public bool Contains(T item)
	{
		if (item is null)
		{
			throw new ArgumentNullException(nameof(item), "Item to check for
existance cannot be null.");
		}

		return IndexOf(item) != -1;
	}

	public void CopyTo(T[] array, int arrayIndex)
	{
		if (array == 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 (count > array.Length - arrayIndex)
            {
                  throw new ArgumentException("The destination array does not have
enough space.");
            }

            LinkedListNode current = head;
            while (current != null)
            {
                  array[arrayIndex++] = current.Data;
                  current = current.Next;
            }
      }

      public int IndexOf(T item)
      {
            if (item == null)
            {
                  throw new ArgumentNullException(nameof(item), "Item to search
cannot be null.");
            }

            int index = 0;
            LinkedListNode current = head;

            while (current != null)
            {
                  if (EqualityComparer<T>.Default.Equals(current.Data, item))
                  {
                        return index;
                  }

                  current = current.Next;
                  index++;
            }

            return -1;
      }

      public void Insert(int index, T item)
      {
            if (index < 0 || index > count)
                  throw new ArgumentOutOfRangeException(nameof(index));

            LinkedListNode newNode = new LinkedListNode(item);

            if (index == 0)
            {
                  newNode.Next = head;
                  head = newNode;
            }
            else
            {
                  LinkedListNode current = head;
                  for (int i = 0; i < index - 1; i++)
                  {
                        current = current.Next;
                  }

                  newNode.Next = current.Next;
                  current.Next = newNode;
            }
```

```
            count++;
            OnItemAdded(item, index);
        }

        public bool Remove(T item)
        {
            if (item == null)
            {
                throw new ArgumentNullException(nameof(item), "Item to search
cannot be null.");
            }


            LinkedListNode current = head;
            LinkedListNode previous = null;

            while (current != null)
            {
                if (EqualityComparer<T>.Default.Equals(current.Data, item))
                {
                    if (previous == null)
                    {
                        head = current.Next;
                    }
                    else
                    {
                        previous.Next = current.Next;
                    }
                    count--;
                    OnItemRemoved(item, Count);
                    return true;
                }

                previous = current;
                current = current.Next;
            }

            return false;
        }

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

            if (index == 0)
                head = head.Next;
            else
            {
                LinkedListNode current = head;
                LinkedListNode previous = null;

                for (int i = 0; i < index; i++)
                {
                    previous = current;
                    current = current.Next;
                }

                previous.Next = current.Next;
                OnItemRemoved(current.Data, Count);
            }

            count--;
        }
```

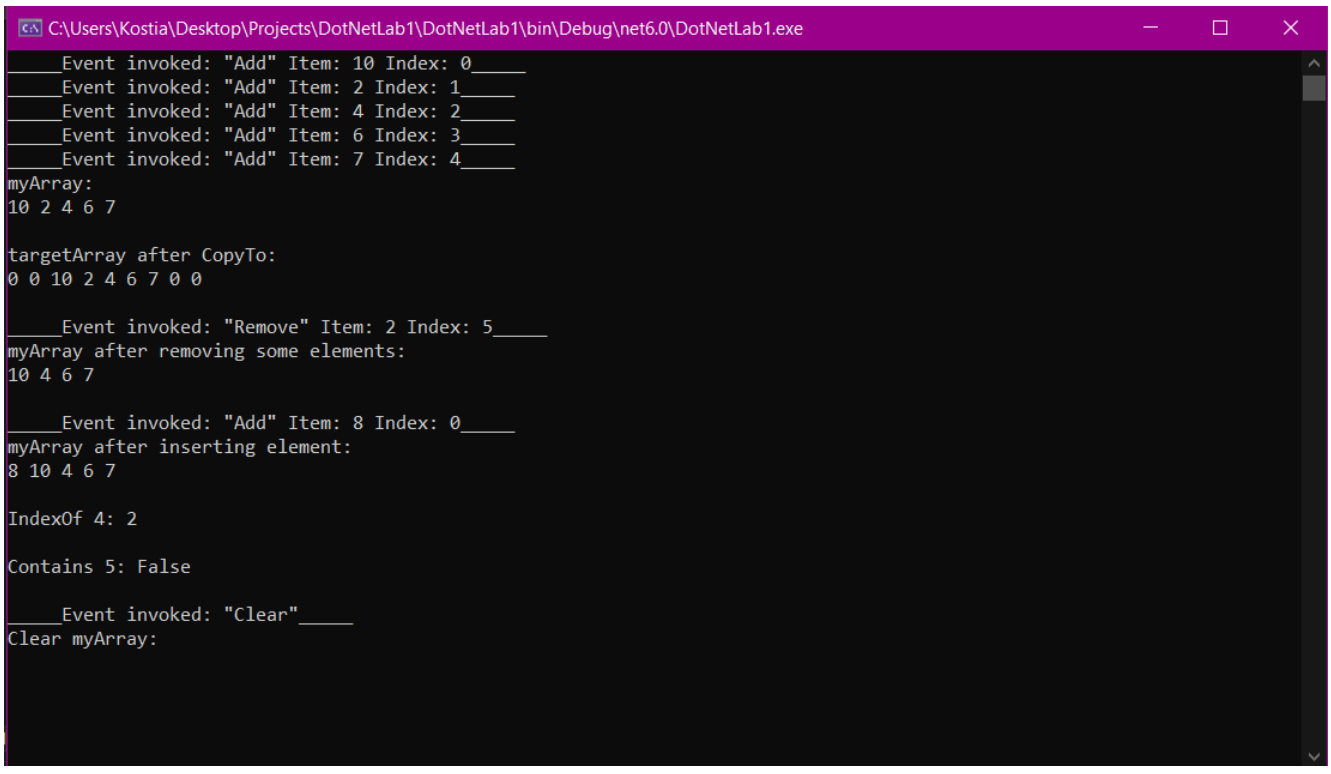
```
        public void Clear()
        {
                head = null;
                count = 0;
                OnArrayCleared();
        }

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

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

Program.cs

```
using DotNetLab1.MyCollections.CustomEventArgs;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using DotNetLab1.MyCollections;

namespace DotNetLab1.Lab1
{
        public static class Program
        {
                public static void PrintEventInvoke(object sender,
ArrayItemEventArgs<int> e)
                {
                        Console.WriteLine($"_____Event invoked: \"{e.Action}\" Item:
{e.Item} Index: {e.Index}_____");
                }
                public static void Foreach<T>(this IEnumerable<T> enumerable, Action<T>
action)
                {
                        foreach (var item in enumerable)
                        {
                                action(item);
                        }
                }
                static void Main(string[] args)
                {

                        DynamicArray<int> myArray = new DynamicArray<int>();

                        myArray.ItemAdded += PrintEventInvoke!;
                        myArray.ItemRemoved += PrintEventInvoke!;
                        myArray.ArrayCleared += (sender, e) =>
Console.WriteLine($"_____Event invoked: \"{e.Action}\"_____");

                        myArray.Add(10);
                        myArray.Add(2);
                        myArray.Add(4);
                        myArray.Add(6);
                        myArray.Add(7);
                        Console.WriteLine("myArray:");
                        myArray.Foreach(item => Console.Write(item + " "));
```

```
                int[] targetArray = new int[9];

                Console.WriteLine("\n");
                myArray.CopyTo(targetArray, 2);
                Console.WriteLine("targetArray after CopyTo:");
                targetArray.Foreach(item => Console.Write(item + " "));

                Console.WriteLine("\n");
                myArray.Remove(3);
                myArray.RemoveAt(1);
                Console.WriteLine("myArray after removing some elements:");
                myArray.Foreach(item => Console.Write(item + " "));

                Console.WriteLine("\n");
                myArray.Insert(0, 8);
                Console.WriteLine("myArray after inserting element:");
                myArray.Foreach(item => Console.Write(item + " "));

                if(myArray.IndexOf(4) == -1)
                        Console.WriteLine("\n\nIndexOf 4: No such element in
array");
                else
                        Console.WriteLine($"\n\nIndexOf 4: {myArray.IndexOf(4)}");

                Console.WriteLine($"\nContains 5: {myArray.Contains(5)}");

                Console.Write("\n");
                myArray.Clear();
                Console.WriteLine("Clear myArray:");
                myArray.Foreach(item => Console.Write(item + " "));

                Console.ReadLine();
            }
        }
}
```

Результат виконання програми:



Висновок:

У ході виконання лабораторної роботи №1:

Була розроблена власна узагальнена колекція під назвою DynamicArray<T>, яка реалізує інтерфейси IEnumerable<T> та IList<T>. Ця колекція має методи для внесення даних будь-якого типу, видалення, пошуку та інших операцій відповідно до типу колекції. Важливо відзначити, що колекція була реалізована з нуля.

До класу DynamicArray<T> була додана підтримка подій. Він має події для відстеження додавання та видалення елементів з колекції, а також подію для відстеження очищення колекції.

Був створений консольний додаток, в якому продемонстровано використання розробленої власної колекції DynamicArray<T>. Додаток підписується на події колекції та демонструє основні операції з колекцією, такі як додавання, видалення, зміна та очищення.