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»

Варіант 9

Виконав студент ІП-12 Єльчанінов Артем Юрійович

Перевірив(ла) Бардін Владислав

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.

Приклади виключних ситуацій: вихід за межи диапазону чи неприпустимий аргумент (індекс), відсутнє значення за ключем/індексом, несумісна зі станом об’єкту операція.

Приклади подій: очищення колекції, додавання, видалення елементу, потрапляння в початок\кінець.

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

Виконання:

MyEventHandlers.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1.MyList {
    public class MyEventHandlers {
        public static void ListCreatedEventHandler(object sender, MyListCreatedEventArgs e)
{
            Console.WriteLine($"\nEvent: array creating; Capacity: {e.Capacity}\n");
        }
        public static void ListResizedEventHandler(object sender, MyListNewSizeEventArgs e)
{
            Console.WriteLine($"\nEvent: change size; OldCapacity: {e.OldCapacity},
NewCapacity: {e.NewCapacity}\n");
        }

        public static void ListItemChangesEventHandler<T>(object sender,
MyListItemChangesEventArgs<T> e) {
            Console.WriteLine($"\nEvent: {e.ItemChangeType}; Item: {e.Item}; Index:
{e.Index}\n");
        }
    }
}
```

MyList.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1.MyList {
    public class MyList<T> : IList<T> {

        private T[] _items;
        private const int DefaultCapacity = 5;

        private int _capacity;
        private int _size;

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

        public EventHandler<MyListNewSizeEventArgs> ArrayResized;
        public EventHandler<MyListCreatedEventArgs> ArrayCreated;
        public EventHandler<MyListItemChangesEventArgs<T>> ItemAdded;
        public EventHandler<MyListItemChangesEventArgs<T>> ItemRemoved;

        public MyList(int capacity = DefaultCapacity)
        {
            if (capacity < 0) {
                throw new ArgumentOutOfRangeException(nameof(capacity), "Capacity is
invalid");
            }
            _capacity = capacity;
            _size = 0;
            _items = capacity is 0 ? Array.Empty<T>() : new T[capacity];
            ArrayCreated += MyEventHandlers.ListCreatedEventHandler!;
            OnArrayCreated();
        }
```

```
        public T this[int index] {
            get => _items[index];
            set {
                if (index < 0 || index >= _items.Length - 1) {
                    throw new ArgumentOutOfRangeException(nameof(index), "Index is out of
range");
                }

                _items[index] = value;
            }
        }

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

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

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

        public void CopyTo(T[] array, int arrayIndex) {
            if(array == null) throw new ArgumentNullException(nameof(array));
            if (array.Length - arrayIndex < _items.Length) {
                throw new ArgumentException("Dest array is too small");
            }
            Array.Copy(_items, 0, array, arrayIndex,_items.Length);
        }

        public int IndexOf(T item) {
            return Array.IndexOf(_items, item);
        }

        public void Insert(int index, T item) {
            if(_size < index || index < 0) {
                throw new ArgumentOutOfRangeException("Inalid index. Index must be in
range.");
            }
            if(_capacity == _size) {
                Resize();
            }
            OnItemAdded(item, index);
            Array.Copy(_items, index, _items, index + 1, _size - index);
            _size++;
            _items[index] = item;
        }

        public bool Remove(T item) {
            if(_size == 0) {
                throw new InvalidOperationException("You can't remove item because array is
empty");
            }
            var index = Array.IndexOf(_items, item);
            var isRemoved = index != -1;
            RemoveAt(index);
            return isRemoved;
        }
```

```
        public void RemoveAt(int index) {
            if (index > _size || index < 0) {
                throw new ArgumentOutOfRangeException("You can't remove item because index
is out of range");
            }
            _size--;
            OnItemRemoved(_items[index], index);
            Array.Copy(_items, index + 1, _items, index, _size - index);
        }

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

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

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

            OnArrayResized(NewCapacity / 2, _capacity);
        }

        private void OnArrayResized(int oldCapacity, int newCapacity) {
            if (ArrayResized != null) {
                ArrayResized(this, new MyListNewSizeEventArgs(oldCapacity, newCapacity));
            }
        }

        private void OnArrayCreated() {
            if (ArrayCreated != null) {
                ArrayCreated(this, new MyListCreatedEventArgs(this._capacity));
            }
        }

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

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

MyListEnumerator.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1.MyList {
    public class MyListEnumerator<T> : IEnumerator<T> {
```

```
        private readonly IList<T> _list;

        private int _index;
        private T _current;
        public T Current => _current;
        object IEnumerator.Current => _current!;

        public MyListEnumerator(IList<T> list) {
            _list = list;
            _index = 0;

            if(list.Any() == true) {
                _current = _list[_index];
            }
            else {
                _current = default!;
            }
        }

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

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

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

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

MyListEventArgs.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab1.MyList {
    public class MyListNewSizeEventArgs : EventArgs {

        public int OldCapacity { get; private set; }
        public int NewCapacity { get; private set; }
        public MyListNewSizeEventArgs(int OldCapacity, int NewCapacity)
        {
            this.OldCapacity = OldCapacity;
            this.NewCapacity = NewCapacity;
        }
    }

    public class MyListCreatedEventArgs : EventArgs {
        public int Capacity { get; private set; }
        public MyListCreatedEventArgs(int Capacity)
        {
            this.Capacity = Capacity;
        }
    }

    public enum ItemChangeType {
        ItemAdded,
```

```
        ItemRemoved
    }

    public class MyListItemChangesEventArgs<T> : EventArgs {
        public ItemChangeType ItemChangeType { get; private set; }

        public T Item { get; private set; }
        public int Index { get; private set; }

        public MyListItemChangesEventArgs(T item, int index, ItemChangeType itemChangeType)
        {
            Item = item;
            this.Index = index;
            ItemChangeType = itemChangeType;
        }
    }
}
```

Program.cs:

```
using MyListLibrary;
namespace Lab1 {
    internal class Program {
        static void Main(string[] args) {
            MyList<int> list;

            list = new MyList<int>(5) { 1, 2, 3, 4, 5 };
            list.ArrayResized += MyEventHandlers.ListResizedEventHandler!;
            list.ItemAdded += MyEventHandlers.ListItemChangesEventHandler!;
            list.ItemRemoved += MyEventHandlers.ListItemChangesEventHandler!;

            list.Add(1);
            OutputMyIntArray(list);

            Console.WriteLine("Array Contains 1? : " + list.Contains(1));

            Console.WriteLine("Getting index 5, result: " + list[5]);
            list[5] = 10;
            Console.WriteLine("Setting 10 in index 5, result: " + list[5]);

            list.Clear();
            OutputMyIntArray(list);

            list = new MyList<int>(5) { 1, 2, 3, 4, 5 };
            list.ArrayResized += MyEventHandlers.ListResizedEventHandler!;
            list.ItemAdded += MyEventHandlers.ListItemChangesEventHandler!;
            list.ItemRemoved += MyEventHandlers.ListItemChangesEventHandler!;

            Console.WriteLine("List contains 20?: " +list.Contains(20));



            Console.WriteLine("Index of 5: " + list.IndexOf(5));
            list.Insert(2, 100);
            Console.Write("List after insert 100 on 10 index: ");
            OutputMyIntArray(list);

            list.Remove(1);

            Console.Write("List after removing 1: ");
            OutputMyIntArray(list);

            list.RemoveAt(3);
            Console.Write("List after removing index 3: ");
            OutputMyIntArray(list);



        }
```

```
        public static void OutputMyIntArray(MyList<int> list) {
            Console.Write("Array: ");
            for (int i = 0; i < list.Count; i++) {
                Console.Write(list[i] + "  ");
            }
            Console.WriteLine();
        }
    }
}
```

Приклад виконання:

```
Event: array creating; Capacity: 5


Event: change size; OldCapacity: 5, NewCapacity: 10


Event: ItemAdded; Item: 1; Index: 5

Array: 1  2  3  4  5  1
Array Contains 1? : True
Getting index 5, result: 1
Setting 10 in index 5, result: 10
Array:

Event: array creating; Capacity: 5

List contains 20?: False
Index of 5: 4

Event: change size; OldCapacity: 5, NewCapacity: 10


Event: ItemAdded; Item: 100; Index: 2

List after insert 100 on 10 index: Array: 1  2  100  3  4  5

Event: ItemRemoved; Item: 1; Index: 0

List after removing 1: Array: 2  100  3  4  5

Event: ItemRemoved; Item: 4; Index: 3

List after removing index 3: Array: 2  100  3  5
```

Посилання на код:

[Web_.Net_Labs/Lab1 at master · LesterGaben/Web_.Net_Labs (github.com)]