МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Лабораторна робота №1

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

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

Виконав:

студент гр. ІС-11

Гіжицький Д. О.

Викладач:

Бардін В.

**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 9:** ****

**Посилання на репозиторій:**

<https://github.com/doomer2004/dotNetLabs2023term1/tree/main/GenericLab1>

**Лістинг коду:**

**CustomEventArgs**

ArrayAction.cs:

public enum ArrayAction

{

Add,

Remove,

Clear,

Resize

}

ArrayEventArgs.cs:

public class ArrayEventArgs : EventArgs

{

public ArrayAction Action { get; private set; }

public TimeOnly ActionTime { get; private set; }

public ArrayEventArgs(ArrayAction action)

{

Action = action;

ActionTime = TimeOnly.FromDateTime(DateTime.Now);

;

}

}

ArrayItemEventArgs.cs

public class ArrayItemEventArgs<T> : ArrayEventArgs

{

public T Item { get; private set; }

public int Index { get; private set; }

public ArrayItemEventArgs(T item, int index, ArrayAction action) : base(action)

{

Item = item;

Index = index;

}

}

ArrayResizedEventArgs.cs

public class ArrayResizedEventArgs

: ArrayEventArgs

{

public int OldCapacity { get; private set; }

public int NewCapacity { get; private set; }

public ArrayResizedEventArgs(int oldCapacity, int newCapacity) : base(ArrayAction.Resize)

{

OldCapacity = oldCapacity;

NewCapacity = newCapacity;

}

}

DynamicList.cs

public sealed class DynamicList<T> : IList<T> where T : struct

{

private const int DefaultCapacity = 4;

private int \_size;

private int \_capacity;

private T[] \_items;

public int Count => \_size;

public bool IsReadOnly { get; } = false;

public DynamicList()

{

\_capacity = DefaultCapacity;

\_items = new T[\_capacity];

}

public DynamicList(int capacity)

{

if (capacity < 0)

{

throw new ArgumentOutOfRangeException(nameof(capacity));

}

else if(capacity > 0)

{

\_capacity = DefaultCapacity;

\_items = new T[\_capacity];

}

else

{

\_capacity = capacity;

\_size = 0;

\_items = Array.Empty<T>();

}

}

public DynamicList(IEnumerable<T> items)

{

if (items is null)

throw new ArgumentNullException(nameof(items));

\_capacity = DefaultCapacity;

\_items = new T[\_capacity];

foreach (var item in items)

{

this.Add(item);

}

}

public EventHandler<ArrayItemEventArgs<T>> ItemAdded;

public EventHandler<ArrayItemEventArgs<T>> ItemRemoved;

public EventHandler<ArrayEventArgs> DynamicListCleared;

public EventHandler<ArrayResizedEventArgs> DynamicListResized;

private void OnItemAdded(T item, int index)

{

if (ItemAdded != null)

{

ItemAdded(this, new ArrayItemEventArgs<T>(item, index, ArrayAction.Add));

}

}

private void OnItemRemoved(T item, int index)

{

if (ItemRemoved != null)

{

ItemRemoved(this, new ArrayItemEventArgs<T>(item, index, ArrayAction.Remove));

}

}

private void OnArrayCleared()

{

if (DynamicListCleared != null)

{

DynamicListCleared(this, new ArrayEventArgs(ArrayAction.Clear));

}

}

private void OnArrayResized(int oldCapacity)

{

if (DynamicListResized != null)

{

DynamicListResized(this, new ArrayResizedEventArgs(oldCapacity, \_capacity));

}

}

public IEnumerator<T> GetEnumerator()

{

return new DynamicListEnumerator<T>(this);

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Add(T item)

{

if (\_size >= \_capacity)

{

Resize();

}

\_items[\_size] = item;

\_size++;

}

public void Clear()

{

\_items = new T[DefaultCapacity];

\_capacity = \_size;

OnArrayCleared();

}

public bool Contains(T item)

{

for (int i = 0; i < \_size; i++)

{

var elements = \_items[i];

if (elements.Equals(item))

{

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, 0,

array, arrayIndex, \_items.Length );

}

public bool Remove(T item)

{

var index = Array.IndexOf(\_items, item);

var isRemoved = index != -1;

RemoveAt(index);

return isRemoved;

}

private void Resize()

{

var oldCapacity = \_capacity;

var newCapacity = \_capacity \* 2;

var tempArray = new T[newCapacity];

Array.Copy(\_items, tempArray, \_size);

\_items = tempArray;

\_capacity = newCapacity;

OnArrayResized(oldCapacity);

}

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 == index)

\_items[index] = item;

if (\_size == \_capacity)

Resize();

int properIndex = GetProperIndex(index, toInsert: true);

\_items[properIndex] = item;

\_size++;

OnItemAdded(item, index);

}

private int GetProperIndex(int index, bool toInsert = false)

{

if (\_size == 0 && !toInsert)

{

throw new ArgumentOutOfRangeException(nameof(index));

}

int count = toInsert ? \_size + 1 : \_size;

return index >= 0 ? index % count : (count + (index % count)) % count;

}

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);

}

public T this[int index]

{

get => \_items[index];

set

{

if (index >= \_size)

{

throw new ArgumentException("Invalid index");

}

\_items[index] = value;

}

}

}

Program.cs

using System.Numerics;

using System.Threading.Channels;

using DynamicList;

using DynamicList.CustomEventArgs;

var lst = new DynamicList<int>(4);

lst.ItemAdded += PrintArrayItemEventArgs!;

lst.ItemRemoved += PrintArrayItemEventArgs!;

lst.DynamicListCleared += (sender, e) => Console.WriteLine($"Event invoked: {e.Action} {e.ActionTime}");

lst.DynamicListResized += (sender, e) =>

Console.WriteLine($"Event invoked: {e.Action} {e.ActionTime} Old capacity: {e.OldCapacity}- New capacity: {e.NewCapacity}");

Console.WriteLine("Insert to array");

lst.Insert(0, 1);

lst.Insert(1, 2);

lst.Insert(2, 3);

lst.Insert(3, 4);

OutputCollection(lst);

Console.WriteLine("Remove from array");

lst.RemoveAt(3);

Console.WriteLine($"Contains check: {lst.Contains(1)}");

Console.WriteLine("Copy to array");

var lst2 = new int[]{5, 6, 7, 8, 9, 10, 0, 0, 0};

lst.CopyTo(lst2 , 4);

OutputCollection(lst);

OutputCollection(lst2);

Console.WriteLine("Clear array");

lst.Clear();

Console.WriteLine("Length of array: " + lst.Count);

Console.ReadLine();

void PrintArrayItemEventArgs(object sender, ArrayItemEventArgs<int> e)

{

Console.WriteLine($"Event invoked: {e.Action} {e.ActionTime} [{e.Item} : {e.Index}]");

}

void OutputCollection(IEnumerable<int> collection)

{

foreach (var item in collection)

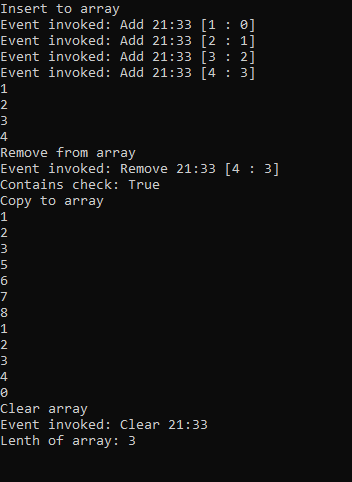
{

Console.WriteLine(item);

}

}

**Консольна програма:**



**Висновок:** в ході роботи я навчився проектувати та реалізовувати узагальнені типи, впровадження підтримку подій для типу.

**Питання до роботи:**

1) Дайте визначання колекції. Як колекція пов’язана з інтерфейсами IEnumerable та ICollection?

Колекція – це набір зв’язаних, однотипних об’єктів динамічного розміру. Всі колекції повинні реалізувати інтерфейс ICollection, який в свою чергу реалізує інтерфейс IEnumerable.

2) Розкажіть про основні інтерфейси необхідні для функціонування колекцій та LINQ.

Перш за все, колекція має імплементувати IEnumerable або IEnumerable<T> (якщо колекція узагальнена) інтерфейс. В інтерфейсі є лише один метод – GetEnumerator, який дозволяє ітерувати елементи колекції.

Для функціонування LINQ колекція повинна реалізовувати IQueryable або IQueryable<T> (якщо колекція узагальнеа) інтерфейс.

3) Розкажіть про призначення і можливі сценарії застосування інтерфейсу IEnumerator.

Інтерфейс IEnumerator потрібен для того, щоб колекцію можна було ітерувати без прямого доступу до неї. Це потрібно, коли ми хочемо задати власну логіку перебору елементів в колекції. Інтерфейс містить метод MoveNext, який відповідає за перехід до наступного елементу, властивість Current для поточного елементу та метод Reset для того, щоб повернутись на початок.

4) Порівняйте інтерфейси IEnumerable та IAsyncEnumerable.

IEnumerable виконується синхронно, IAsyncEnumerable – асинхронно.

5) Розкажіть про призначення і обмеження generic типів.

Generic методи були створені для того, щоб програмісти могли працювати з різними типами даних, при цьому не дублюючи код. Ми створюємо загальне рішення, яке потім можемо використовувати з різними типами без необхідності створювати окремий код для кожного різного типу. Обмеження для generic типів були додані для того, щоб звузити об’єм типів для узагальненого методу, класу, тощо.

6) Поясніть призначення оператору default та його обмеження.

Опетор default відповідає за те, щоб повернути значення за замовчуванням залежно від типу. Наприклад, int – 0, string – null, тощо. Для того, щоб використовувати default, тип повинен мати значення за замовчуванням, тобто клас, інтерфейс не зможуть бути використані для цього. (Якщо вони не позначені як Nullable)

7) Розкажіть про лямбда вирази. Наведіть приклад використання лямбда-виразу.

Лямбда-вираз – це скорочений запис анонімних функцій або такі делегати, де нам не потрібно визначати окремий метод.

8) Розкрийте різниці між expression & statement лямбдами.

Перш за все, вони відрізняються виразами, які виконують та як повертають значення. Expression лямбди завжди повертають одне значення, коли statement взагалі не повертає значення. Прикладом Expression лямбди може бути делегат Func, тоді, коли для statement – Action.

9) Розкажіть про події, і як вони реалізовані в С#.

Подія – це механізм, при якому всі об’єкти можуть підписатись, отримати сповіщення та зреагувати на нього. Події реалізують патерн Observer. Визначення події базується на основі делегату.

10) Поясніть, яким чином виконується підписання на події та скасування підписки.

Для підписки чи скасування події потрібен екземпляр делегату, який буде відповідати сигнатурі події.

Підписка на подію відбувається за допомогою оператора +=, коли скасування підписки за допомогою -=.

11) Наведіть склад класу делегату та поясніть, чим забезпечується контроль типів в делегатах.

Делегат – це тип, який представляє собою посилання на метод. Вони зберігають та викликають методи. Контроль типів забезпечується сигнатурою делегату, тому сигнатура методу має відповідати сигнатурі делегату.