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»

Виконав:

студент гр. ІС-11

Бацан Ілля

Викладач:

Бардін В.

2023 рік

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

**Завдання:**

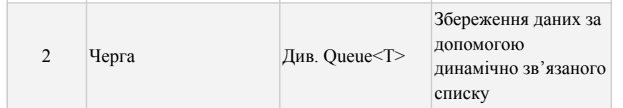
1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та 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 2:**



**Посилання на код GitHub:**

https://github.com/UkrRozvidka/.Net-Lab1-Queue  
Код основного класу:

**Queue:**

namespace Queue

{

public class Queue<T> : ICollection, IReadOnlyCollection<T>

{

#region fields and properties

private int count;

private QueueNode<T>? \_head;

private QueueNode<T>? \_tail;

public int Count

{

get { return count; }

}

public QueueNode<T>? First

{

get { return \_head; }

}

public QueueNode<T>? Last

{

get { return \_tail; }

}

#endregion

#region events

public delegate void QueueHandler(object sender, QueueEventArgs<T> e);

public event QueueHandler? OnAddElement;

public event QueueHandler? OnRemoveElement;

#endregion

#region constructors

public Queue() { }

public Queue(IEnumerable<T> collection)

{

if (collection == null)

{

throw new ArgumentNullException(nameof(collection));

}

foreach (T item in collection)

{

Enqueue(item);

}

}

#endregion

#region basic methods

// Adds a new item to the tail of the queue

public void Enqueue(T item)

{

if (item == null)

throw new ArgumentNullException(nameof(item));

var newNode = new QueueNode<T>(item);

if (count == 0)

{

\_head = newNode;

\_tail = newNode;

}

else

{

newNode.Prev = \_tail;

newNode.Next = null;

\_tail!.Next = newNode;

\_tail = newNode;

}

count++;

// Trigger the event to notify listeners that an element was added

OnAddElement?.Invoke(this, new QueueEventArgs<T>(newNode.Value, "element was added to tail"));

}

// Removes and returns the element at the head of the queue.

public T Dequeue()

{

if (count == 0)

throw new InvalidOperationException("empty");

var oldHeadValue = \_head!.Value;

if (count == 1)

{

\_head = null;

\_tail = null;

}

else

{

\_head!.Next!.Prev = null;

\_head = \_head.Next;

}

count--;

// Trigger the event to notify listeners that an element was removed

OnRemoveElement?.Invoke(this, new QueueEventArgs<T>(oldHeadValue, "element was removed from head"));

return oldHeadValue;

}

public void Clear()

{

QueueNode<T>? current = \_head;

while (current is not null)

{

current = current.Next;

}

\_head = null;

count = 0;

}

public bool Contains(T item)

{

if (item == null)

throw new ArgumentNullException(nameof(item));

foreach(var t in this)

{

if(t!.Equals(item)) return true;

}

return false;

}

// Retrieves the value of the element at the head of the queue without removing it

public T Peek()

{

if (count == 0)

throw new InvalidOperationException("queue is empty");

return \_head!.Value;

}

// Attempts to retrieve the value of the element at the head of the queue without removing it

// Returns true if successful, false if the queue is empty

public bool TryPeek(out T? result)

{

if (count == 0)

{

result = default;

return false;

}

result = \_head!.Value;

return true;

}

// Attempts to remove and return the element at the head of the queue

// Returns true if successful, false if the queue is empty

public bool TryDequeue(out T? result)

{

if (count == 0)

{

result = default;

return false;

}

result = this.Dequeue();

return true;

}

#endregion

#region enumerable and enumerator

public IEnumerator<T> GetEnumerator()

{

QueueNode<T>? current = \_head;

while (current is not null)

{

yield return current.Value;

current = current.Next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

#endregion

public void CopyTo(Array array, int index)

{

if (array == null)

throw new ArgumentNullException(nameof(array));

if (index < 0 || index >= array.Length)

throw new ArgumentOutOfRangeException(nameof(index));

if (array.Length - index < count)

throw new ArgumentException("The destination array has not space to copy the elements.", nameof(array));

if (array.Rank != 1)

throw new ArgumentException("Multidimensional arrays are not supported.", nameof(array));

var current = \_head;

while (current != null)

{

array.SetValue(current.Value, index++);

current = current.Next;

}

}

bool ICollection.IsSynchronized

{

get { return false; }

}

object ICollection.SyncRoot => this;

}

public sealed class QueueNode<T>

{

public QueueNode(T value)

{

Value = value;

}

public QueueNode<T>? Prev { get; internal set; }

public QueueNode<T>? Next { get; internal set; }

public T Value { get; internal set; }

}

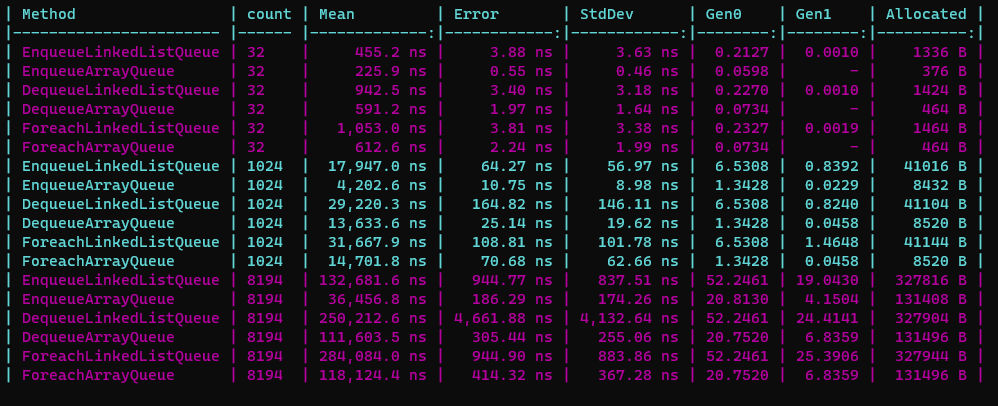
}

Також як додаткове завдання було написання бенчмарків за допомогою бібліотеки BenchmarkDotNet, де порівнювалась моя реалізація черги та стандартна реалізація від Microsoft.

У результаті маємо такі результати:



Generic type – Lazy<Int32>



Generic type – Lazy<Int32>

**Висновки:** в ході роботи я навчився проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.