Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №1

з дисципліни "Сучасні технології розробки WEB-застосувань на платформі .NET"

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

Варіант: №8 Кільцевий список

Виконав:
студент групи ІП-93
Домінський Валентин
Олексійович

Перевірила:
Крамар Юлія Михайлівна

Київ 2022

Зміст:

Мета:

Навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій

Вихідний код

CircularLinkedListNode:

```
namespace Lab1.CircularLinkedListNode;

public class CircularLinkedListNode<T>
{
    public T Data;
    public CircularLinkedListNode<T>? Next;

    public CircularLinkedListNode(T data)
    {
        Data = data;
        Next = null;
    }
}
```

CircularLinkedList:

```
using System.Collections;
using System.Text;
using Lab1.CircularLinkedListNode;

namespace Lab1.CircularLinkedList;

public class CircularLinkedList<T> : ICollection<T>, IEnumerable<T>, ICloneable
{
    #region Fields

    public CircularLinkedListNode<T>? Head { get; private set; }
    public CircularLinkedListNode<T>? Tail { get; private set; }

    public int Count { get; private set; }

    public bool IsReadOnly => false;

    public event Action Added;
    public event Action Removed;

    #endregion Fields

    #region Constructors

    public CircularLinkedList()
    {
        Head = null;
        Tail = null;
    }

    public CircularLinkedList(T item)
    {
        SetFirstElement(item);
    }
```

```
#endregion Constructors

#region Methods

public T this[int index]
{
    get
    {
        CheckCorrectIndex(index);

        var current = GetNodeInRange(Head, index);

        return current.Data;
    }
    set
    {
        CheckNull(value);

        CheckCorrectIndex(index);

        var current = GetNodeInRange(Head, index);

        current.Data = value;
    }
}

public void Add(T item)
{
    CheckNull(item);

    if (IsEmpty())
    {
        SetFirstElement(item);
        return;
    }

    Tail = new CircularLinkedListNode<T>(item);
    Tail.Next = Head;

    var current = GetNodeInRange(Head, Count - 1);

    current.Next = Tail;

    Count++;
    Added?.Invoke();
}

public void AddFirst(T item)
{
    CheckNull(item);

    if (IsEmpty())
    {
        SetFirstElement(item);
        return;
    }

    Head = new CircularLinkedListNode<T>(item)
    {
        Next = Head
    };

    Count++;
    SetTail();
    Added?.Invoke();
}
```

```
    public void AddAt(T item, int index)
    {
        CheckNull(item);
        CheckCorrectIndex(index);

        var current = GetNodeInRange(Head, index - 1);

        var next = current.Next;

        var nodeToInsert = new CircularLinkedListNode<T>(item);

        current.Next = nodeToInsert;
        nodeToInsert.Next = next;

        Count++;
        Added?.Invoke();
    }

    public void Clear()
    {
        Head = Tail = null;
        Count = 0;
        Removed?.Invoke();
    }

    public bool Contains(T item)
    {
        CheckNull(item);

        var current = Head;

        for (var i = 0; i < Count; i++)
        {
            if (Compare(current.Data, item))
            {
                return true;
            }

            current = current.Next;
        }

        return false;
    }

    // regular "==" only works when T is constrained to be a reference type
    // Without any constraints, you can compare with null, but only null - and
    // that comparison will always be false for non-nullable value types.
    private static bool Compare<T>(T x, T y) => EqualityComparer<T>.Default.Equals(x,
y);

    public void CopyTo(T[] array, int arrayIndex)
    {
        var node = Head;

        for (var i = arrayIndex; i < Count; i++)
        {
            array[arrayIndex + i] = node.Data;
            node = node.Next;
        }
    }

    public bool Remove(T item)
    {
        CheckNull(item);

        var current = Head;

        for (var i = 0; i < Count; i++)
```

```
    {
        if (Compare(current.Data, item))
        {
            return RemoveAt(i);
        }

        current = current.Next;
    }

    return false;
}

public void RemoveAll(T item)
{
    CheckNull(item);

    for (int i = Count; i > 0; i--)
    {
        Remove(item);
    }
}

public bool RemoveAt(int index)
{
    CheckCorrectIndex(index);

    var current = Head;
    var previous = Tail;

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

    previous.Next = current.Next;

    ChangeEdgeNodes(previous, index);

    Count--;

    Removed?.Invoke();
    return true;
}

public void RemoveHead()
{
    RemoveAt(0);
}

public void RemoveTail()
{
    RemoveAt(Count - 1);
}

public IEnumerator<T> GetEnumerator()
{
    var node = Head;

    for (var i = 0; i < Count; i++)
    {
        yield return node.Data;
        node = node.Next;
    }
}

IEnumerator IEnumerable.GetEnumerator()
{
```

```
    return GetEnumerator();
}

private static CircularLinkedListNode<T> GetNodeInRange
    (CircularLinkedListNode<T> startingNode, int position)
{
    for (int i = 0; i < position; i++)
    {
        startingNode = startingNode.Next;
    }

    return startingNode;
}

private bool CheckCorrectIndex(int index)
{
    if (index <= Count - 1)
    {
        return true;
    }

    throw new ArgumentOutOfRangeException(nameof(index));
}

private void SetFirstElement(T item)
{
    CheckNull(item);

    var node = new CircularLinkedListNode<T>(item);

    Head = node;
    Head.Next = node;
    Tail = node;
    Tail.Next = node;

    Count++;
    Added?.Invoke();
}

private bool IsEmpty()
{
    if (Count == 0)
    {
        return true;
    }

    return false;
}

private static void CheckNull(T item)
{
    if (item == null)
    {
        throw new ArgumentNullException(nameof(item));
    }
}

private void SetTail()
{
    var currentNode = Head;

    for (var i = 0; i < Count; i++)
    {
        if (i == Count - 1)
        {
            Tail = currentNode;
        }
```

```
            currentNode = currentNode.Next;
        }

        Tail.Next = Head;
    }

    private void ChangeEdgeNodes(CircularLinkedListNode<T> previous, int index)
    {
        if (index == 0)
        {
            Head = previous.Next;
        }
        else if (index == Count - 1)
        {
            Tail = previous;
        }
    }

    public override string ToString()
    {
        StringBuilder list = new();

        var current = Head;

        for (int i = 0; i < Count; i++)
        {
            list.Append(current.Data.ToString());
            list.Append(Environment.NewLine + Environment.NewLine);

            current = current.Next;
        }

        return list.ToString();
    }

    public object Clone()
    {
        CircularLinkedList<T> list = new();
        var current = Head;

        for (int i = 0; i < Count; i++)
        {
            list.Add(current.Data);
            current = current.Next;
        }

        return list;
    }

    #endregion Methods
}
```

Car:

```
namespace Labs;

public class Car
{
    private readonly string _name = "Default name";
    private readonly int _ownerId;
    private readonly string _color = "Default color";

    public Car(int ownerId, string name, string color)
    {
        _ownerId = ownerId;
        _name = name;
        _color = color;
```

```
    }

    public override string ToString()
    {
        var properties = $"Owner Id - {_ownerId}, name - {_name} and color -
{_color}";

        return properties;
    }
}
```

Program:

```
using Lab1.CircularLinkedList;

namespace Labs;

internal class Program
{
    private static CircularLinkedList<Car> s_listing = new();

    private static void Main()
    {
        Car firstCar = new(1, "Toyota", "Red");
        Car secondCar = new(2, "Nissan", "White");
        Car thirdCar = new(3, "Honda", "Black");

        Car[] cars = new Car[] { firstCar, secondCar, thirdCar };

        s_listing.Added += OnAdded;
        s_listing.Removed += OnRemoved;

        RunAllExamples(cars);
    }

    private static void RunAllExamples(Car[] cars)
    {
        AddExample(cars);
        RemoveExample(cars);
        ForeachExample(cars);
        OtherFeaturesExample(cars);
    }

    private static void OnAdded()
    {
        Console.WriteLine("New element added");
        Console.WriteLine(s_listing.ToString());
    }

    private static void OnRemoved()
    {
        Console.WriteLine("Element removed");
        Console.WriteLine(s_listing.ToString());
    }

    private static void AddExample(Car[] cars)
    {
        s_listing.Add(cars[0]);

        // Toyota

        s_listing.AddFirst(cars[1]);

        // Nissan Toyota

        s_listing.AddAt(cars[2], 1);
```

```
    // Nissan Honda Toyota

    ResetList();
}

private static void RemoveExample(Car[] cars)
{
    s_listing.Add(cars[2]);
    s_listing.Add(cars[0]);
    s_listing.Add(cars[0]);
    s_listing.Add(cars[0]);
    s_listing.Add(cars[1]);
    s_listing.Add(cars[2]);

    // Honda Toyota Toyota Toyota Nissan Honda

    s_listing.RemoveAt(2);

    // Honda Toyota Toyota Nissan Honda

    s_listing.Remove(cars[1]);

    // Honda Toyota Toyota Honda

    s_listing.RemoveHead();

    // Toyota Toyota Honda

    s_listing.RemoveTail();

    // Toyota Toyota

    s_listing.RemoveAll(cars[0]);

    // Toyota
    // -_-
}

private static void ForeachExample(Car[] cars)
{
    s_listing.Add(cars[0]);
    s_listing.Add(cars[1]);
    s_listing.Add(cars[2]);

    // Toyota Nissan Honda

    foreach (var car in s_listing)
    {
        Console.WriteLine(car);
    }

    Console.WriteLine("//////");

    foreach (var car in s_listing.Reverse())
    {
        Console.WriteLine(car);
    }
    Console.WriteLine();

    ResetList();
}

private static void OtherFeaturesExample(Car[] cars)
{
    s_listing.Add(cars[0]);
    s_listing.Add(cars[1]);
    s_listing.Add(cars[2]);
```

```
        // Toyota Nissan Honda

        Console.WriteLine(s_listing.Contains(cars[0])); // True;
        Console.WriteLine("//////");

        CircularLinkedList<Car> carClone = (CircularLinkedList<Car>)s_listing.Clone();
        Console.WriteLine(carClone.ToString());

        // Toyota Nissan Honda

        Car[] carsCopy = new Car[s_listing.Count];
        s_listing.CopyTo(carsCopy, 0);

        s_listing.Clear();

        Console.WriteLine(s_listing.ToString()); // -_-
        Console.WriteLine("//////");

        Console.WriteLine(carsCopy[0]); // Toyota
        Console.WriteLine("//////");

        carsCopy[0] = carsCopy[1];

        Console.WriteLine(carsCopy[0]); // Nissan
        Console.WriteLine();

        ResetList();
    }

    private static void ResetList()
    {
        s_listing.Clear();
    }
}
```

Результат роботи:

```
New element added
Owner Id - 3, name - Honda and color - Black

Owner Id - 1, name - Toyota and color - Red

Owner Id - 1, name - Toyota and color - Red

Owner Id - 1, name - Toyota and color - Red

Owner Id - 2, name - Nissan and color - White

Owner Id - 3, name - Honda and color - Black


Element removed
Owner Id - 3, name - Honda and color - Black

Owner Id - 1, name - Toyota and color - Red

Owner Id - 1, name - Toyota and color - Red

Owner Id - 2, name - Nissan and color - White

Owner Id - 3, name - Honda and color - Black
```

Контрольні питання:

1) Дайте визначення колекції. Наведіть типи колекцій
    a. Об'єкт, який зберігає в собі дані
    b. ArrayList, Dictionary, List
2) Наведіть основні інтерфейси, які успадковуються колекціями, та їх призначення
    a. ICollection – базовий функціонал додавання, вилучення елементів
    b. IEnumerable – перерахування
    c. ICloneable – клонування
3) Поясніть призначення паттерну «Ітератор» та його реалізацію в .Net
    a. Ітератор повертає упорядковану послідовність елементів
    b. Для цього використовуються оператор yield
4) Дані якого формату зберігаються у хеш-таблицях, словниках? Які переваги їх використання?
    a. Зберігає ключ-значення
        i. Перший – non-generic, другий – generic
5) Поясніть призначення узагальнених типів. Наведіть приклади.
    a. За допомогою них можна робити колекції з різними (навіть власними) типами. List, Dictionary
6) Поясніть призначення оператору default
    a. Повертає значення за замовченням для певного типу
7) Поясніть призначення обмежень where в узагальнених типах. Наведіть приклади
    a. Дає можливість використовувати лише певні типи (за посиланням, за значенням).
8) Поясніть сутність нумератору колекції
    a. Допомагає при переборі колекції
9) Наведіть способи опису нумератору в колекції, приклад створення власного нумератору
    a. Простий спосіб – це проходження по усіх елементах по одному та повертанню кожного з них. Якщо Вам

треба більш специфічний нумератор, то є методи MoveNext, Reset і т.д.

10) Роз’ясніть сутність поняття делегата
    a. Делегати – гнучкі методи, функціонал яких можна заміняти на льоту

11) Наведіть приклад опису делегата та виклику методу, використовуючи делегат
    a.

```
private delegate void Msg();

private static void Main()
{
  Trigger();
}

private static void Trigger()
{
    Msg message;
    message = Greet;
    message();
}

private static void Greet()
{
    Console.WriteLine("Greet");
}
```

12) Наведіть склад класу делегату та поясніть, чим забезпечується контроль типів в делегатах
    a. –

13) Поясніть сутність поняття анонімного методу
    a. Анонімні типи дозволяють створити об'єкт із деяким набором властивостей без визначення класу

14) Поясніть сутність лямбда-виразу, наведіть приклади лямбда-виразів
    a. Вираз, що повертає метод
    b. x => x * x

15) Наведіть приклад опису події та генерування події
    a. Створюємо делегат (або використовуємо існуючий)
    b. Створюємо подію
    c. Викликаємо подію

16) Поясніть, яким чином виконується підписання на події та скасування підписки
    a. За допомогою «+=» та «-=»

Висновки:

Я дізнався більше інформації створення власних типів колекції та реалізації інтерфейсів, познайомився з делегатами, подіями