

**Міністерство освіти і науки України  
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-застосунків на платформі Microsoft.NET»

Виконав:

студент групи ІМ-12

Кривенок Максим Геннадійович

варіант: 6

Перевірила:

Крамар Ю. М.

Київ 2023

## 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:

6	Словник	Див. Dictionary<TKey, TValue>	Збереження даних за допомогою динамічно зв'язаного списку або вектору
---	---------	-------------------------------------	---

## Хід виконання роботи

Я виконав лабораторну роботу з теми "Розробка узагальненої колекції з підтримкою подій" за варіантом 6.

### 1. Розробка класу власної узагальненої колекції

Для реалізації власної узагальненої колекції я створив клас Dictionary. Цей клас реалізує інтерфейс IDictionary і містить поля head і tail, в яких зберігаються перший і останній елементи колекції.

Для внесення даних в колекцію я реалізував метод Add, який приймає ключ і значення. Для видалення елемента з колекції я реалізував метод Remove, який приймає ключ елемента, який необхідно видалити. Для пошуку елемента за ключем я реалізував метод ContainsKey. Також я реалізував всі інші методи які містить інтерфейс IDictionary<> та додав декілька своїх таких, як Backwards та Insert.

### 2. Додавання до класу власної узагальненої колекції підтримки подій та обробки виключних ситуацій

Додавання до класу події було одним з найлегших етапів виконання лабораторної роботи. Я додав в тіло класу декілька основних подій, а також додав в тіло відповідних методів виклик події у відповідності до призначення події.

### 3. Опис класу колекції та всіх необхідних для роботи з колекцією типів збереження у динамічній бібліотеці

Опис класу колекції і всіх необхідних для роботи з колекцією типів я зберіг у динамічній бібліотеці. Для цього я створив проект Visual Studio C# і додав до нього всі необхідні файли. Потім я зібрав проект і отримав динамічну бібліотеку.

### 4. Створення консольного додатку, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції

Для демонстрації використання розробленої власної колекції я створив консольний додаток. У цьому додатку я реалізував наступні функції:

Внесення даних в колекцію (всі можливі методи)

Видалення елемента з колекції (всі можливі методи)

Пошук елемента за ключем (всі можливі методи)

Підписка на події колекції

Перевіра працездатності виключень.

## Код

### • MyDictionary.cs

```
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;

namespace DotNet_Lab1
{
    public class MyDictionary<TKey, TValue> : IDictionary<TKey, TValue>
        where TKey : notnull
    {
        private class Node
        {
            public KeyValuePair<TKey, TValue> Value { get; set; }
            public Node? Next { get; set; }
            public Node? Prev { get; set; }

            public Node(KeyValuePair<TKey, TValue> value)
            {
                Value = value;
            }
        }

        private Node? head;
        private Node? tail;

        public TValue this[TKey key]
        {
            get
            {

```

```

        if (key is null)
            throw new ArgumentNullException(nameof(key), "The key cannot be
null.");

        Node result = TryGetNode(key, false) ??
            throw new KeyNotFoundException("The given key was not present in the
dictionary");

        return result.Value.Value;
    }
    set
    {
        if (key is null)
            throw new ArgumentNullException(nameof(key), "The key cannot be
null.");

        Node? result = TryGetNode(key, false);
        if (result is not null)
        {
            var oldValue = result.Value;
            result.Value = new KeyValuePair<TKey, TValue>(key, value);

            ElementChanged?.Invoke(this, oldValue, result.Value);
        }
        else
        {
            Add(key, value);
        }
    }
}

public ICollection<TKey> Keys => this.Select(kvPair => kvPair.Key).ToList();

public ICollection<TValue> Values => this.Select(kvPair =>
kvPair.Value).ToList();

public int Count { get; private set; }

public bool IsReadOnly => false;

public event Action<object>? CollectionCleared;
public event Action<object, KeyValuePair<TKey, TValue>[]>? CollectionCopied;

public event Action<object, KeyValuePair<TKey, TValue>>? ElementAdded;

public event Action<object, KeyValuePair<TKey, TValue>>? ElementRemoved;
public event Action<object, KeyValuePair<TKey, TValue>, KeyValuePair<TKey,
TValue>>? ElementChanged;

public void Add(TKey key, TValue value)
{
    Insert(key, value, Count);
}

public void Add(KeyValuePair<TKey, TValue> item)
{
    Insert(item.Key, item.Value, Count);
}

public void Insert(TKey key, TValue value, int index)
{
    if (key is null)
        throw new ArgumentNullException(nameof(key), "The key cannot be null.");

```

```

        if (index < 0 || index > Count)
            throw new ArgumentOutOfRangeException(nameof(index), "The array index is
out of the valid range.");

        if (ContainsKey(key))
            throw new ArgumentException("An item with the same key has already been
added", nameof(key));

        var keyValuePair = new KeyValuePair<TKey, TValue>(key, value);

        var node = new Node(keyValuePair);
        if (index == 0)
        {
            InsertAtBeginning(node);
        }
        else if (index == Count)
        {
            InsertAtEnd(node);
        }
        else
        {
            InsertInMiddle(node, index);
        }

        Count++;
        ElementAdded?.Invoke(this, keyValuePair);
    }

    public void Clear()
    {
        head = null;
        tail = null;

        Count = 0;
        CollectionCleared?.Invoke(this);
    }

    public bool Contains(KeyValuePair<TKey, TValue> item)
    {
        var node = TryGetNode(item.Key, true, item.Value);

        return node is not null;
    }

    public bool ContainsKey(TKey key)
    {
        if (key is null)
            throw new ArgumentNullException(nameof(key), "The key cannot be null.");

        var node = TryGetNode(key, false);

        return node is not null;
    }

    public void CopyTo(KeyValuePair<TKey, TValue>[] array, int arrayIndex)
    {
        if (array is null)
            throw new ArgumentNullException(nameof(array), "The array cannot be
null.");

        if (arrayIndex < 0 || arrayIndex >= array.Length)
            throw new ArgumentOutOfRangeException(nameof(arrayIndex), "The array
index is out of the valid range.");
    }

```

```

        if (array.Length - arrayIndex < Count)
            throw new ArgumentException("The destination array has fewer elements
than the collection.");

        Node? node = head;
        int i = arrayIndex;

        while (node is not null)
        {
            array[i++] = node.Value;
            node = node.Next;
        }

        CollectionCopied?.Invoke(this, array);
    }

    public IEnumerator<KeyValuePair<TKey, TValue>> GetEnumerator()
    {
        Node? node = head;
        while (node is not null)
        {
            yield return node.Value;
            node = node.Next;
        }
    }

    public IEnumerable<KeyValuePair<TKey, TValue>> Backwards()
    {
        Node? node = tail;
        while (node is not null)
        {
            yield return node.Value;
            node = node.Prev;
        }
    }

    public bool Remove(TKey key)
    {
        if (key is null)
            throw new ArgumentNullException(nameof(key), "The key cannot be null.");

        var removedNode = TryRemoveNode(key, false);

        if (removedNode is null)
        {
            return false;
        }

        return true;
    }

    public bool Remove(KeyValuePair<TKey, TValue> item)
    {
        var removedNode = TryRemoveNode(item.Key, true, item.Value);

        if (removedNode is null)
        {
            return false;
        }

        return true;
    }

    public bool TryGetValue(TKey key, [MaybeNullWhen(false)] out TValue value)

```

```

{
    if (key is null)
        throw new ArgumentNullException(nameof(key), "The key cannot be null.");

    value = default;

    var node = TryGetNode(key, false);
    bool result = node is not null;

    if (result)
    {
        value = node!.Value.Value;
    }

    return result;
}

private Node? TryGetNode(TKey key, bool isValuePassed, TValue? value = default)
{
    Node? node = head;
    while (node is not null)
    {
        if (node.Value.Key.Equals(key) &&
            (!isValuePassed ||
             (node.Value.Value?.Equals(value) ?? value is null) ) )
        {
            return node;
        }
        node = node.Next;
    }

    return null;
}

private Node? TryRemoveNode(TKey key, bool isValuePassed, TValue? value =
default)
{
    var nodeToRemove = TryGetNode(key, isValuePassed, value);

    if (nodeToRemove is null)
        return null;

    Node? previousNode = nodeToRemove.Prev;
    Node? nextNode = nodeToRemove.Next;

    if (previousNode is not null)
    {
        previousNode.Next = nextNode;
    }

    if (nextNode is not null)
    {
        nextNode.Prev = previousNode;
    }

    if (nodeToRemove.Equals(head))
    {
        head = nodeToRemove.Next;
    }

    if (nodeToRemove.Equals(tail))
    {
        head = nodeToRemove.Prev;
    }
}

```

```

        Count--;
        ElementRemoved?.Invoke(this, nodeToRemove.Value);

        return nodeToRemove;
    }

    private void InsertAtBeginning(Node node)
    {
        node.Next = head;

        if (head is not null)
        {
            head.Prev = node;
        }

        head = node;
        tail ??= node;
    }

    private void InsertAtEnd(Node node)
    {
        node.Prev = tail;

        if (tail is not null)
        {
            tail.Next = node;
        }

        tail = node;
        head ??= node;
    }

    private void InsertInMiddle(Node node, int index)
    {
        Node current = head!;
        for (int i = 0; i < index; i++)
        {
            current = current.Next!;
        }

        Node previousNode = current.Prev!;
        Node nextNode = current;

        previousNode.Next = node;

        nextNode.Prev = node;

        node.Prev = previousNode;
        node.Next = nextNode;
    }

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

```

## - Program.cs

```

using DotNet_Lab1;
using System.ComponentModel;

```



```

var myDictionary = new MyDictionary<string, string>();

myDictionary.CollectionCleared += col => Console.WriteLine("CollectionCleared");
myDictionary.CollectionCopied += (col, _) => Console.WriteLine("CollectionCopied");
myDictionary.ElementAdded += (col, _) => Console.WriteLine("ElementAdded");
myDictionary.ElementRemoved += (col, _) => Console.WriteLine("ElementRemoved");
myDictionary.ElementChanged += (col, _, _) => Console.WriteLine("ElementChanged");

myDictionary.Add("2", "2");
PrintAllElements(".Add(\"2\", \"2\")", myDictionary);

myDictionary.Add(new KeyValuePair<string, string>("3", "3"));
PrintAllElements(".Add(new KeyValuePair<string, string>(\"3\", \"3\"))", myDictionary);

myDictionary.Insert("1", "1", 0);
PrintAllElements(".Insert(\"1\", \"1\", 0)", myDictionary);

myDictionary.Insert("5", "5", 3);
PrintAllElements(".Insert(\"5\", \"5\", 3)", myDictionary);

myDictionary.Insert("4", "4", 3);
PrintAllElements(".Insert(\"4\", \"4\", 3)", myDictionary);

Console.WriteLine($"Count - {myDictionary.Count}\n");

myDictionary.Remove("3");
PrintAllElements(".Remove(\"3\")", myDictionary);

myDictionary.Remove(new KeyValuePair<string, string>("2", "1"));
PrintAllElements(".Remove(new KeyValuePair<string, string>(\"2\", \"1\"))",
myDictionary);

string? stringResult;
var boolResult = myDictionary.TryGetValue("5", out stringResult);
Console.WriteLine($"TryGetValue(\"5\", out stringResult): {boolResult}, Value =
{stringResult}\n");

bool isContainsKey = myDictionary.ContainsKey("3");
Console.WriteLine($"ContainsKey(\"3\"): {isContainsKey}\n");

bool isContainsKeyValue = myDictionary.Contains(new KeyValuePair<string, string>("4",
"4"));
Console.WriteLine($"ContainsKey(\"3\"): {isContainsKey}\n");

myDictionary["1"] = "2";
PrintAllElements("[\"1\"] = \"2\";", myDictionary);

Console.WriteLine($"[\"4\"]: {myDictionary["4"]}\n");

try
{
    Console.WriteLine("[\"999\"]");
    var test = myDictionary["999"];
}
catch(Exception ex)
{
    Console.WriteLine(ex.Message + '\n');
}

try
{
    Console.WriteLine(".Add(\"2\", \"2\");");
    myDictionary.Add("2", "2");
}

```

```

}
catch (Exception ex)
{
    Console.WriteLine(ex.Message + '\n');
}

myDictionary.Clear();
PrintAllElements(".Clear()", myDictionary);

void PrintAllElements(string operation, MyDictionary<string, string> myDictionary)
{
    Console.WriteLine($"{operation}:");
    foreach (var item in myDictionary)
    {
        Console.WriteLine($"{item.Key} - {item.Value}");
    }
    Console.WriteLine();
}

```

## Висновок

Під час виконання лабораторної роботи я оволодів навичками проектування та реалізації узагальнених типів, а також типів з підтримкою подій. Завдяки розробці власної узагальненої колекції яка використовувала динамічно зв'язаний список для зберігання даних, я здобув розуміння роботи інтерфейсів колекцій в бібліотеках System.Collections та System.Collections.Generic, а також навчився їх використовувати для створення власних реалізацій колекцій.