Міністерство освіти і науки України КПІ ім. Ігоря Сікорського

Факультет Інформатики та Обчислювальної Технiки

**ЗВІТ**

**Лабораторна робота №1 з дисципліни**

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

|  |  |
| --- | --- |
| Перевірив:  Викладач кафедри ІСТ  ФІОТ  Бардін В. | Виконав:  Хрисанфов Дмитро  гр. ІК-13 |

2023

**Узагальнені типи (Generic) з підтримкою подій. Колекції**

**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.

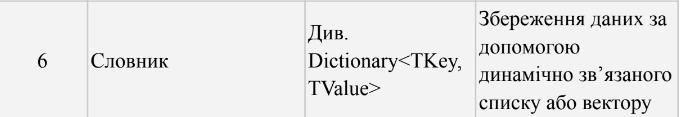
3. Опис класу колекції та всіх необхідних для роботи з колекцією типів

зберегти у динамічній бібліотеці.

4. Створити консольний додаток, в якому продемонструвати

використання розробленої власної колекції, підписку на події

колекції.



Код програми

public class MyDictionary<TKey, TValue> : IEnumerable

{

private int size = 100;

private Item<TKey, TValue>[] Items;

private List<TKey> Keys = new List<TKey>();

public event Action<TKey, TValue> ItemAdded;

public event Action<TKey> ItemRemoved;

public event Action DictionaryCleared;

public MyDictionary()

{

Items = new Item<TKey, TValue>[size];

ItemAdded += (key, value) => Console.WriteLine($"Element with key {key} and value {value} added.");

ItemRemoved += key => Console.WriteLine($"Element with key {key} is removed.");

DictionaryCleared += () => Console.WriteLine("Dictionary is cleared.");

}

public void Add(Item<TKey, TValue> item)

{

var hash = GetHash(item.Key);

if (Keys.Contains(item.Key))

{

return;

}

if (Items[hash] == null)

{

Keys.Add(item.Key);

Items[hash] = item;

ItemAdded?.Invoke(item.Key, item.Value);

}

else

{

var placed = false;

for (var i = hash; i < size; i++)

{

if (Items[i] == null)

{

Keys.Add(item.Key);

Items[i] = item;

placed = true;

ItemAdded?.Invoke(item.Key, item.Value);

break;

}

if (Items[i].Key.Equals(item.Key))

{

return;

}

}

if (!placed)

{

for (var i = 0; i < hash; i++)

{

if (Items[i] == null)

{

Keys.Add(item.Key);

Items[i] = item;

placed = true;

ItemAdded?.Invoke(item.Key, item.Value);

break;

}

if (Items[i].Key.Equals(item.Key))

{

return;

}

}

}

if (!placed)

{

throw new Exception("Out of dictionary range");

}

}

}

public void Remove(TKey key)

{

var hash = GetHash(key);

if (!Keys.Contains(key))

{

return;

}

if (Items[hash] == null)

{

for (var i = 0; i < size; i++)

{

if (Items[i] != null && Items[i].Key.Equals(key))

{

Items[i] = null;

Keys.Remove(key);

ItemRemoved?.Invoke(key);

return;

}

}

return;

}

if (Items[hash].Key.Equals(key))

{

Items[hash] = null;

Keys.Remove(key);

ItemRemoved?.Invoke(key);

}

else

{

var placed = false;

for (var i = hash; i < size; i++)

{

if (Items[i] == null)

{

return;

}

if (Items[i].Key.Equals(key))

{

Items[i] = null;

Keys.Remove(key);

ItemRemoved?.Invoke(key);

return;

}

}

if (!placed)

{

for (var i = 0; i < hash; i++)

{

if (Items[i] == null)

{

return;

}

if (Items[i].Key.Equals(key))

{

Items[i] = null;

Keys.Remove(key);

ItemRemoved?.Invoke(key);

return;

}

}

}

}

}

public TValue Search(TKey key)

{

var hash = GetHash(key);

if (!Keys.Contains(key))

{;

return default(TValue);

}

if (Items[hash] == null)

{

foreach (var item in Items)

{

if (item.Key.Equals(key))

{

return item.Value;

}

}

return default(TValue);

}

if (Items[hash].Key.Equals(key))

{

return Items[hash].Value;

}

else

{

var placed = false;

for (var i = hash; i < size; i++)

{

if (Items[i] == null)

{

return default(TValue);

}

if (Items[i].Key.Equals(key))

{

return Items[i].Value;

}

}

if (!placed)

{

for (var i = 0; i < hash; i++)

{

if (Items[i] == null)

{

return default(TValue);

}

if (Items[i].Key.Equals(key))

{

return Items[i].Value;

}

}

}

}

return default(TValue);

}

public bool ContainsKey(TKey key)

{

foreach (var existingKey in Keys)

{

if (existingKey.Equals(key))

{

return true;

}

}

return false;

}

public bool ContainsValue(TValue value)

{

foreach (var item in Items)

{

if (item != null && item.Value.Equals(value))

{

return true;

}

}

return false;

}

public bool TryGetValue(TKey key, out TValue value)

{

var hash = GetHash(key);

if (Keys.Contains(key))

{

if (Items[hash] == null)

{

foreach (var item in Items)

{

if (item != null && item.Key.Equals(key))

{

value = item.Value;

return true;

}

}

}

else if (Items[hash].Key.Equals(key))

{

value = Items[hash].Value;

return true;

}

else

{

for (var i = hash; i < size; i++)

{

if (Items[i] == null)

{

break;

}

if (Items[i].Key.Equals(key))

{

value = Items[i].Value;

return true;

}

}

for (var i = 0; i < hash; i++)

{

if (Items[i] == null)

{

break;

}

if (Items[i].Key.Equals(key))

{

value = Items[i].Value;

return true;

}

}

}

}

value = default(TValue);

return false;

}

public bool TryAdd(Item<TKey, TValue> item)

{

if (!Keys.Contains(item.Key))

{

Add(item);

return true;

}

return false;

}

public IEnumerator GetEnumerator()

{

foreach (var item in Items)

{

if (item != null)

{

yield return item;

}

}

}

private int GetHash(TKey key)

{

return key.GetHashCode() % size;

}

public void Clear()

{

var keysCopy = new List<TKey>(Keys);

foreach (var key in keysCopy)

{

Remove(key);

}

DictionaryCleared?.Invoke();

}

}

public class Item<TKey, TValue>

{

public TKey Key { get; set; }

public TValue Value { get; set; }

public Item(TKey key, TValue value)

{

Key = key;

Value = value;

}

public override int GetHashCode()

{

return Key.GetHashCode();

}

public override string ToString()

{

return Value.ToString();

}

}

using Lab1;

using System;

namespace Lab1

{

public class Program

{

static void Main(string[] args)

{

var dict = new MyDictionary<int, string>();

dict.TryAdd(new Item<int, string>(1, "One"));

dict.Add(new Item<int, string>(1, "One"));

dict.Add(new Item<int, string>(2, "Two"));

dict.Add(new Item<int, string>(3, "Three"));

dict.Add(new Item<int, string>(4, "Four"));

dict.Add(new Item<int, string>(5, "Five"));

dict.Add(new Item<int, string>(101, "One hundred and one"));

foreach (var item in dict)

{

Console.WriteLine(item);

}

Console.WriteLine(dict.Search(7));

Console.WriteLine(dict.Search(3));

Console.WriteLine(dict.Search(101));

dict.Remove(7);

dict.Remove(3);

dict.Remove(1);

dict.Remove(101);

foreach (var item in dict)

{

Console.WriteLine(item);

}

dict.Clear();

Console.WriteLine();

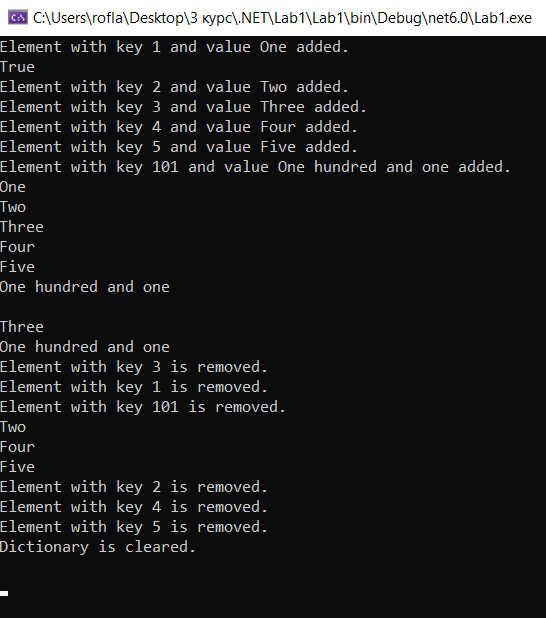
Console.ReadLine();

}

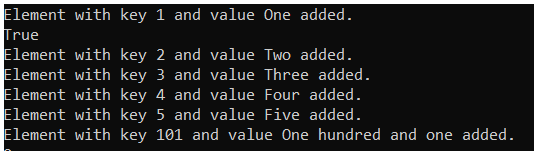
}

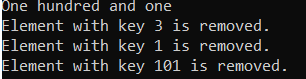
}

Результати роботи програми



Методи Add/TryAdd/Remove та підтримка подій:

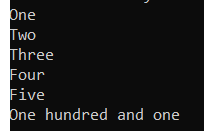




Методи Clear:



Робота циклу foreach



Обробка виключних ситуацій, в разі видалення елементу, ключа якого не існує

