Міністерство освіти і науки України КПІ ім. Ігоря Сікорського

Факультет Інформатики та Обчислювальної Техніки

ЗВІТ

Лабораторна робота №2 з дисципліни

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

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

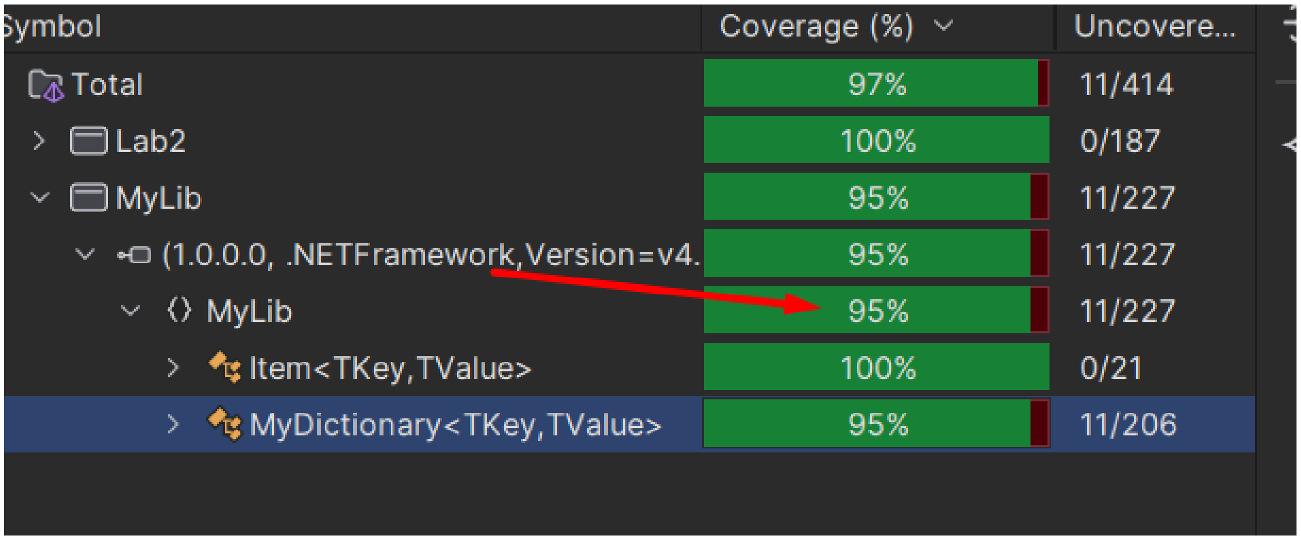
Лабораторна робота 2

Модульне тестування. Ознайомлення з засобами та практиками модульного тестування

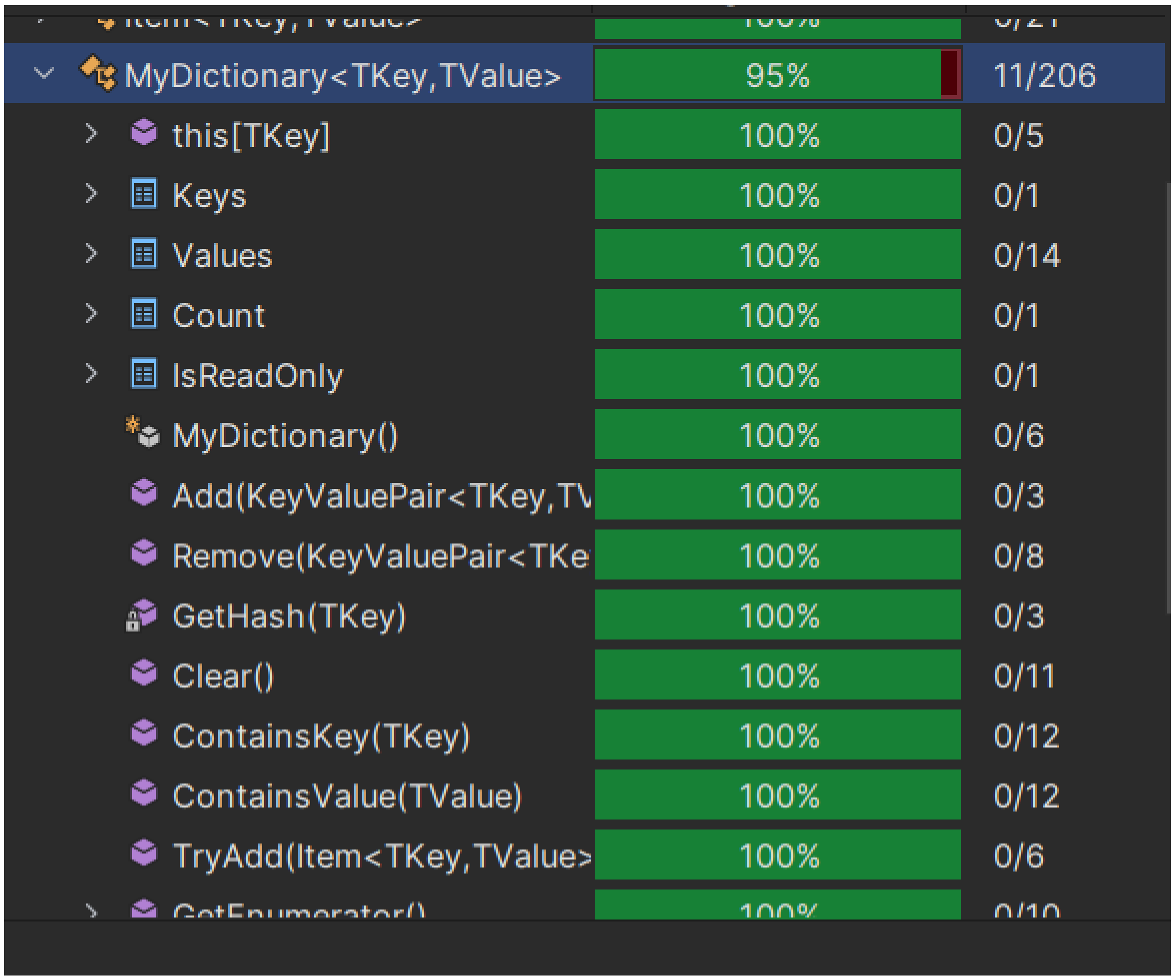
Мета лабораторної роботи – навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

Завдання:
1. Додати до проекту власної узагальненої колекції (застосувати виконану лабораторну роботу No1) проект модульних тестів, використовуючи певний фреймворк (Nunit, Xunit, тощо).
2. Розробити модульні тести для функціоналу колекції.
3. Дослідити ступінь покриття модульними тестами вихідного коду колекції, використовуючи, наприклад, засіб AxoCover.

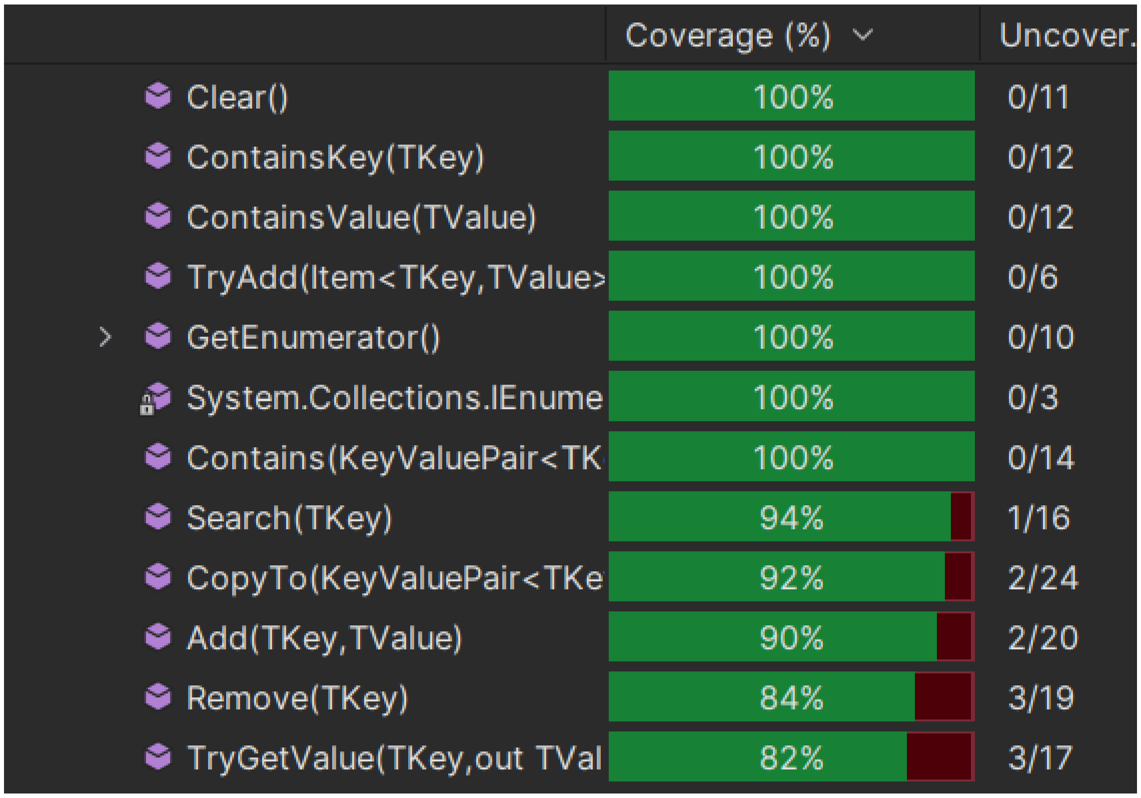
Результат роботи програми:



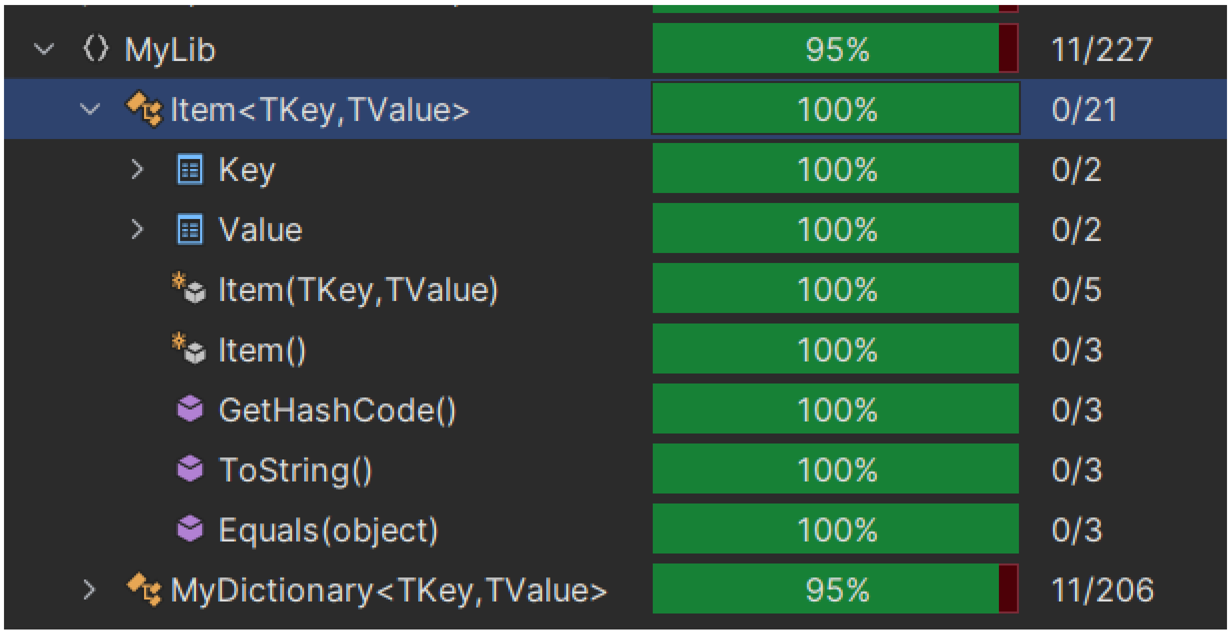
Symbol	Coverage (%)	Uncovere...
Total	97%	11/414
Lab2	100%	0/187
MyLib	95%	11/227
(1.0.0.0, .NETFramework,Version=v4.	95%	11/227
MyLib	95%	11/227
Item<TKey,TValue>	100%	0/21
MyDictionary<TKey,TValue>	95%	11/206



Item<TKey,TValue>	100%	0/21
MyDictionary<TKey,TValue>	95%	11/206
this[TKey]	100%	0/5
Keys	100%	0/1
Values	100%	0/14
Count	100%	0/1
IsReadOnly	100%	0/1
MyDictionary()	100%	0/6
Add(KeyValuePair<TKey,TV	100%	0/3
Remove(KeyValuePair<TKe	100%	0/8
GetHash(TKey)	100%	0/3
Clear()	100%	0/11
ContainsKey(TKey)	100%	0/12
ContainsValue(TValue)	100%	0/12
TryAdd(Item<TKey,TValue>	100%	0/6
GetEnumerator()	100%	0/10

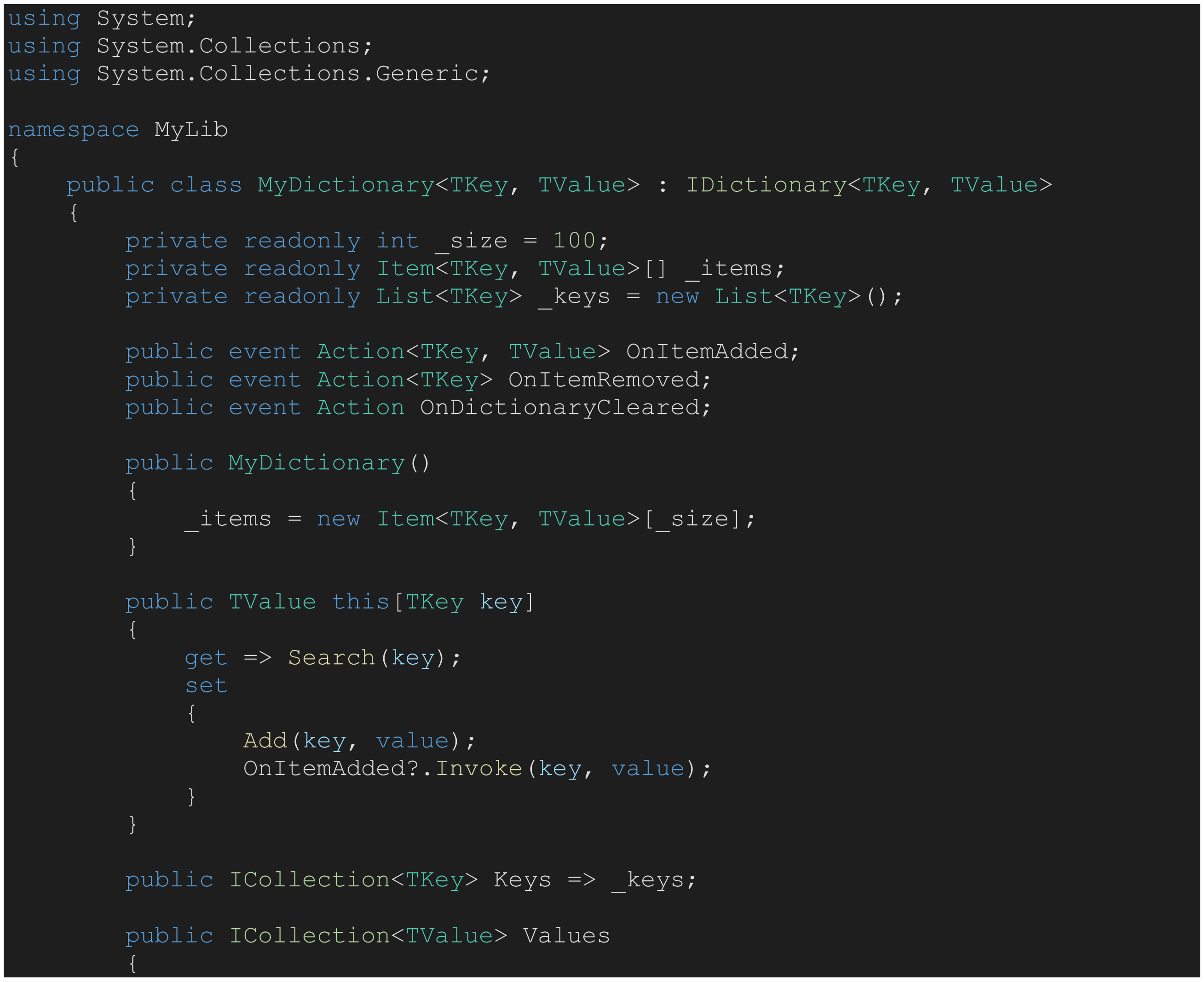


	Coverage (%)	Uncover.
Clear()	100%	0/11
ContainsKey(TKey)	100%	0/12
ContainsValue(TValue)	100%	0/12
TryAdd(Item<TKey,TValue>	100%	0/6
GetEnumerator()	100%	0/10
System.Collections.IEnume	100%	0/3
Contains(KeyValuePair<TK	100%	0/14
Search(TKey)	94%	1/16
CopyTo(KeyValuePair<TKe	92%	2/24
Add(TKey,TValue)	90%	2/20
Remove(TKey)	84%	3/19
TryGetValue(TKey,out TVal	82%	3/17



MyLib	95%	11/227
Item<TKey,TValue>	100%	0/21
Key	100%	0/2
Value	100%	0/2
Item(TKey,TValue)	100%	0/5
Item()	100%	0/3
GetHashCode()	100%	0/3
ToString()	100%	0/3
Equals(object)	100%	0/3
MyDictionary<TKey,TValue>	95%	11/206

Код програми:



```
using System;
using System.Collections;
using System.Collections.Generic;

namespace MyLib
{
    public class MyDictionary<TKey, TValue> : IDictionary<TKey, TValue>
    {
        private readonly int _size = 100;
        private readonly Item<TKey, TValue>[] _items;
        private readonly List<TKey> _keys = new List<TKey>();

        public event Action<TKey, TValue> OnItemAdded;
        public event Action<TKey> OnItemRemoved;
        public event Action OnDictionaryCleared;

        public MyDictionary()
        {
            _items = new Item<TKey, TValue>[_size];
        }

        public TValue this[TKey key]
        {
            get => Search(key);
            set
            {
                Add(key, value);
                OnItemAdded?.Invoke(key, value);
            }
        }

        public ICollection<TKey> Keys => _keys;

        public ICollection<TValue> Values
        {
            get
            {
                List<TValue> values = new List<TValue>();
                foreach (var item in _items)
                {
                    if (item != null)
                    {
                        values.Add(item.Value);
                    }
                }
                return values;
            }
        }

        public int Count => _keys.Count;

        public bool IsReadOnly => false;

        public TValue Search(TKey key)
        {
            if (_keys.Contains(key))
            {
                var hash = GetHash(key);
                for (var i = 0; i < _size; i++)
                {
                    var index = (hash + i) % _size;
                    if (_items[index] != null && _items[index].Key.Equals(key))
                    {
                        return _items[index].Value;
                    }
                }
            }

            throw new KeyNotFoundException($"Key '{key}' not found in the
dictionary.");
        }



        public void Add(TKey key, TValue value)
        {
            var hash = GetHash(key);

            if (hash < 0)
            {
                throw new ArgumentException($"Hash value for key '{key}' is less
than 0.");
            }

            if (_keys.Contains(key))
            {
                throw new ArgumentException($"Key '{key}' already exists in the
dictionary.");
            }

            int index = hash;
            while (_items[index] != null)
            {
                index = (index + 1) % _size;
                if (index == hash)
                {
                    throw new Exception("Out of dictionary range");
                }
            }

            _keys.Add(key);
            _items[index] = new Item<TKey, TValue> { Key = key, Value = value };
            OnItemAdded?.Invoke(key, value);
        }



        public bool Remove(TKey key)
        {
            var hash = GetHash(key);

            if (!_keys.Contains(key))
            {
                throw new InvalidOperationException($"Key '{key}' not found.");
            }

            for (var i = 0; i < _size; i++)
            {
                var index = (hash + i) % _size;
                if (_items[index] != null && _items[index].Key.Equals(key))
                {
                    _items[index] = null;
                    _keys.Remove(key);
                    OnItemRemoved?.Invoke(key);
                    return true;
                }
            }

            throw new InvalidOperationException($"Key '{key}' not found.");
        }



        public bool TryGetValue(TKey key, out TValue value)
        {
            value = default;

            var hash = GetHash(key);

            if (!_keys.Contains(key))
            {
                throw new InvalidOperationException($"Key '{key}' not found.");
            }

            int index = hash;
            while (_items[index] != null)
            {
                if (_items[index].Key.Equals(key))
                {
                    value = _items[index].Value;
                    return true;
                }
                index = (index + 1) % _size;
            }

            throw new InvalidOperationException($"Key '{key}' not found.");
        }



        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);
            }

            OnDictionaryCleared?.Invoke();
        }

        public bool ContainsKey(TKey key)
        {
            foreach (var item in _items)
            {
                if (item != null && item.Key.Equals(key))
                {
                    return true;
                }
            }

            throw new KeyNotFoundException($"$Key '{key}' not found.");
        }

        public bool ContainsValue(TValue value)
        {
            foreach (var item in _items)
            {
                if (item != null &&
EqualityComparer<TValue>.Default.Equals(item.Value, value))
                {
                    return true;
                }
            }

            throw new Exception($"Value '{value}' not found.");
        }

        public void TryAdd(Item<TKey, TValue> item)
        {
            if (_keys.Contains(item.Key))
            {
                throw new InvalidOperationException($"Key '{item.Key}' already
exists.");
            }
            Add(item.Key, item.Value);
        }

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

        public bool Remove(KeyValuePair<TKey, TValue> item)
        {
            if (_keys.Contains(item.Key))
            {
                Remove(item.Key);
                return true;
            }
            else
            {
                throw new Exception("Item not found.");
            }
        }

        public IEnumerator<KeyValuePair<TKey, TValue>> GetEnumerator()
        {
            foreach (var key in _keys)
            {
                var hash = GetHash(key);
                yield return new KeyValuePair<TKey, TValue>(key,
_items[hash].Value);
            }
        }

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

        public bool Contains(KeyValuePair<TKey, TValue> item)
        {
            foreach (var key in _keys)
            {
                var hash = GetHash(key);
                var currentItem = _items[hash];

                if (currentItem != null && currentItem.Key.Equals(item.Key) &&
EqualityComparer<TValue>.Default.Equals(currentItem.Value, item.Value))
                {
                    return true;
                }
            }

            throw new Exception("Item not found.");
        }

        public void CopyTo(KeyValuePair<TKey, TValue>[] array, int arrayIndex)
        {
            if (array == null)
            {
                throw new ArgumentNullException(nameof(array));
            }
            if (arrayIndex < 0 || arrayIndex >= array.Length)
            {
                throw new ArgumentOutOfRangeException(nameof(arrayIndex));
            }

            if (array.Length - arrayIndex < _keys.Count)
            {
                throw new ArgumentException("The destination array is not large
enough.", nameof(array));
            }

            int i = arrayIndex;
            foreach (var key in _keys)
            {
                var hash = GetHash(key);
                var currentItem = _items[hash];

                if (currentItem != null)
                {
                    array[i++] = new KeyValuePair<TKey, TValue>(currentItem.Key,
currentItem.Value);
                }
            }
        }


    }
}
namespace MyLib
{
    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 Item() { }

        public override int GetHashCode()
        {
            return Key.GetHashCode();
        }

        public override string ToString()
        {
            return Value.ToString();
        }

        public override bool Equals(object obj)
        {
            return base.Equals(obj);
        }
    }
}
```

Юніт тести:

```
using MyLib;
public class AddTests
{
    [Theory]
    [InlineData(0, "hey2")]
    [InlineData(250, "hey3")]
    [InlineData(26.1, "hey4")]
    [InlineData(101, null)]
    public void ValidItemAdded(int key, string value)
    {
        MyDictionary<int, string> dict = new MyDictionary<int, string>();
        dict.Add(key, value);
        Assert.False(dict.IsReadOnly);
    }

    [Theory]
    [InlineData(1, "hey2")]
    public void ExistingItemAdded(int key, string value)
    {
        var dict = new MyDictionary<int, string>(){};
        dict.Add(key, value);
        Assert.Throws<ArgumentException>(() =>  dict.Add(key, value));
    }

    [Theory]
    [InlineData(-10, "hey2")]
    public void NegativeItemAdded(int key, string value)
    {
        var dict = new MyDictionary<int, string>(){};
        Assert.Throws<ArgumentException>(() => dict.Add(key,value));
    }


    [Theory]
    [InlineData(1, "Value1")]
    public void AddKeyValuePair_AddsItemForNonExistentKey(int key, string value)
    {
        var dictionary = new MyDictionary<int, string>();
        var item = new KeyValuePair<int, string>(key, value);

        dictionary.Add(item);

        Assert.Equal(value, dictionary[1]);
    }
}
```

```
using MyLib;
public class ClearTests
{
    [Theory]
    [InlineData(0, "hey2")]
    [InlineData(250, "hey3")]
    [InlineData(26.1, "hey4")]
    [InlineData(26.1, "hey4")]
    public void ClearDictionary(int key, string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key,value);

        dictionary.Clear();

        Assert.Empty(dictionary);
    }
}
```

```
using MyLib;

public class ContainsTests
{
    [Theory]
    [InlineData(1, "Value1")]
    public void ContainsKey_ReturnsTrueForExistingKey(int key,string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var result = dictionary.ContainsKey(1);

        Assert.True(result);
    }

    [Theory]
    [InlineData(1, "Value1")]
    public void Contains_ReturnsTrueForExistingKeyValuePair(int key,string
value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var result = dictionary.Contains(new KeyValuePair<int, string>(key,
value));

        Assert.True(result);
    }

    [Theory]
    [InlineData(1, "Value1","Value2")]
    public void Contains_ReturnsFalseForNonExistentKeyValuePair(int key,string
value,string value2)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        Assert.Throws<Exception>(()=>dictionary.Contains(new KeyValuePair<int,
string>(key, value2)));
    }

    [Theory]
    [InlineData(1, "Value1")]
    public void ContainsValue_ReturnsTrueForExistingValue(int key,string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var result = dictionary.ContainsValue(value);

        Assert.True(result);
    }

    [Theory]
    [InlineData(1, "Value1", "Value2")]
    public void ContainsValue_ReturnsFalseForNonExistentValue(int key,string
value,string value2)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        Assert.Throws<Exception>(() => dictionary.ContainsValue(value2));
    }
}
```

```
using MyLib;

public class CopyToTests
{
    [Theory]
    [InlineData(1, "Value1")]
    [InlineData(2, "Value2")]
    [InlineData(3, "Value3")]
    public void CopyTo_CopiesElementsToArrayStartingAtIndex(int key,string
value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);
        var array = new KeyValuePair<int, string>[5];

        dictionary.CopyTo(array, 2);

        Assert.Equal(default(KeyValuePair<int, string>), array[0]); // Первые
два элемента должны быть пустыми
        Assert.Equal(new KeyValuePair<int, string>(key, value), array[2]);
    }

    [Theory]
    [InlineData(1, "Value1")]
    public void CopyTo_ThrowsArgumentNullExceptionForNullArray(int key, string
value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);
        var array = default(KeyValuePair<int, string>[]);

        Assert.Throws<ArgumentNullException>(() => dictionary.CopyTo(array, 0));
    }

    [Theory]
    [InlineData(1, "Value1", -1)]
    public void
CopyTo_ThrowsArgumentOutOfRangeExceptionForNegativeArrayIndex(int key, string
value,int negIndex)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);
        var array = new KeyValuePair<int, string>[1];

        Assert.Throws<ArgumentOutOfRangeException>(() =>
dictionary.CopyTo(array, negIndex));
    }
}
```

```
using MyLib;

public class CountTests
{
    [Fact]
    public void Count_ReturnsZeroForEmptyDictionary()
    {
        var dictionary = new MyDictionary<int, string>();

        var count = dictionary.Count;

        Assert.Equal(0, count);
    }
}
```

```
using MyLib;

public class GetEnumeratorTests
{
    [Theory]
    [InlineData(1, "Value1")]
    public void GetEnumerator_ReturnsValidEnumerator(int key, string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var enumerator = dictionary.GetEnumerator();

        int count = 0;
        while (enumerator.MoveNext())
        {
            count++;
        }

        int expected = dictionary.Count();

        Assert.Equal(expected, count);
    }
}
```

```
global using Xunit;
```

```
using MyLib;

public class IndexerTests
{
    [Theory]
    [InlineData(1, "Value1")]
    public void Indexer_SetValueByKey_AddsNewKeyValueToDictionary(int key,string
value)
    {
        var dictionary = new MyDictionary<int, string>();

        dictionary[key] = value;

        Assert.Equal(value, dictionary[key]);
    }
}
```

```
using MyLib;
using Xunit;

public class ItemTests
{
    [Theory]
    [InlineData(42, "TestValue")]
    public void ItemConstructor_SetsKeyAndValue(int key, string value)
    {
        var item = new Item<int, string>(key, value);

        Assert.Equal(key, item.Key);
        Assert.Equal(value, item.Value);
    }

    [Theory]
    [InlineData(42, "TestValue")]
    public void GetHashCode_ReturnsKeyHashCode(int key, string value)
    {
        var item = new Item<int, string>(key, value);

        int hashCode = item.GetHashCode();

        Assert.Equal(key.GetHashCode(), hashCode);
    }

    [Theory]
    [InlineData(42, "TestValue")]
    public void ToString_ReturnsValueString(int key, string value)
    {
        var item = new Item<int, string>(key, value);

        string stringValue = item.ToString();

        Assert.Equal(value, stringValue);
    }

    [Theory]
    [InlineData(42, "TestValue")]
    public void Equals_ReturnsFalseForEqualItems(int key, string value)
    {
        var item1 = new Item<int, string>(key, value);
        var item2 = new Item<int, string>(key, value);

        bool areEqual = item1.Equals(item2);

        Assert.False(areEqual);
    }
}
```

```
using MyLib;

public class RemoveUnitTests
{
    [Fact]
    public void Remove_ThrowsExceptionForNonExistentKey()
    {
        var dictionary = new MyDictionary<int, string>();

        Assert.Throws<InvalidOperationException>(() => dictionary.Remove(1));
    }

    [Theory]
    [InlineData(1, "Value1")]
    public void
RemoveKeyValuePair_ReturnsTrueAndRemovesItemForExistingKeyValuePair(int
key,string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var result = dictionary.Remove(new KeyValuePair<int, string>(key,
value));

        Assert.True(result);
        Assert.Throws<KeyNotFoundException>(() =>
            Assert.False(dictionary.ContainsKey(key)));
    }

    [Fact]
    public void RemoveKeyValuePair_ReturnsFalseForNonExistentKeyValuePair()
    {
        // Arrange
        var dictionary = new MyDictionary<int, string>();
        Assert.Throws<Exception>(()=>dictionary.Remove(new KeyValuePair<int,
string>(1, "Value1")));
    }
}
```

```
using MyLib;

public class SearchTests
{
    [Theory]
    [InlineData(1, "Value1",101, "Value2")]
    public void Search_ReturnsValueForCollidingKeys(int key, string value, int
key2, string  value2)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);
        dictionary.Add(key2, value2);

        var value3 = dictionary.Search(101);

        Assert.Equal(value2, value3);
    }

    [Theory]
    [InlineData(1, 101,"Value1")]
    public void Search_ThrowsKeyNotFoundExceptionForCollidingNonExistentKey(int
key, int key2, string value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key,value);

        Assert.Throws<KeyNotFoundException>(() => dictionary.Search(key2));
    }
}
```

```
using MyLib;

public class TryAddTests
{
    [Theory]
    [InlineData(1, "Value1")]
    public void TryAdd_AddsItemForNonExistentKey(int key, string value)
    {
        var dictionary = new MyDictionary<int, string>();
        var item = new Item<int, string> { Key = key, Value = value };

        dictionary.TryAdd(item);

        Assert.Equal(value, dictionary[key]);
    }

    [Theory]
    [InlineData(1, "Value1","Value2")]
    public void TryAdd_ThrowsExceptionForExistingKey(int key, string
value,string value2)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key,value);
        var item = new Item<int, string> { Key = key, Value = value2 };

        Assert.Throws<InvalidOperationException>(() => dictionary.TryAdd(item));
    }
}
```

```
using MyLib;

public class TryGetValueTests
{
    [Theory]
    [InlineData(1, "Value1")]
    public void TryGetValue_ReturnsTrueAndSetsValueForExistingKey(int key,string
value)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key, value);

        var result = dictionary.TryGetValue(1, out var value2);

        Assert.True(result);
        Assert.Equal(value, value2);
    }

    [Fact]
    public void TryGetValue_ThrowsInvalidOperationExceptionForNonExistentKey()
    {
        var dictionary = new MyDictionary<int, string>();

        Assert.Throws<InvalidOperationException>(() => dictionary.TryGetValue(1,
out var value));
    }
}
```

```
using MyLib;

public class ValuesTests
{
    [Theory]
    [InlineData(1, "Value1",2, "Value2",3, "Value3")]
    public void Values_ReturnsCorrectCollectionOfValues(int key1, string
value1,int key2, string value2,int key3, string value3)
    {
        var dictionary = new MyDictionary<int, string>();
        dictionary.Add(key1, value1);
        dictionary.Add(key2, value2);
        dictionary.Add(key3, value3);

        var values = dictionary.Values;

        Assert.Equal(new[] { value1, value2, value3 }, values);
    }
}
```

Висновок:

Протягом виконання лабораторної роботи я дослідив фреймворк xUnit і навчився писати юніт тести до своєї власно створеної колекції.