МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»

Кафедра інформаційних систем та технологій

Лабораторна робота №2

з дисципліни «Розробка програмного забезпечення на платформі .Net»

на тему:

«Модульне тестування. Ознайомлення з засобами та практиками модульного тестування»

Виконав:

Студент групи ІС-12

Канупа Максим

Викладачк:
Бардін В.

Київ – 2023

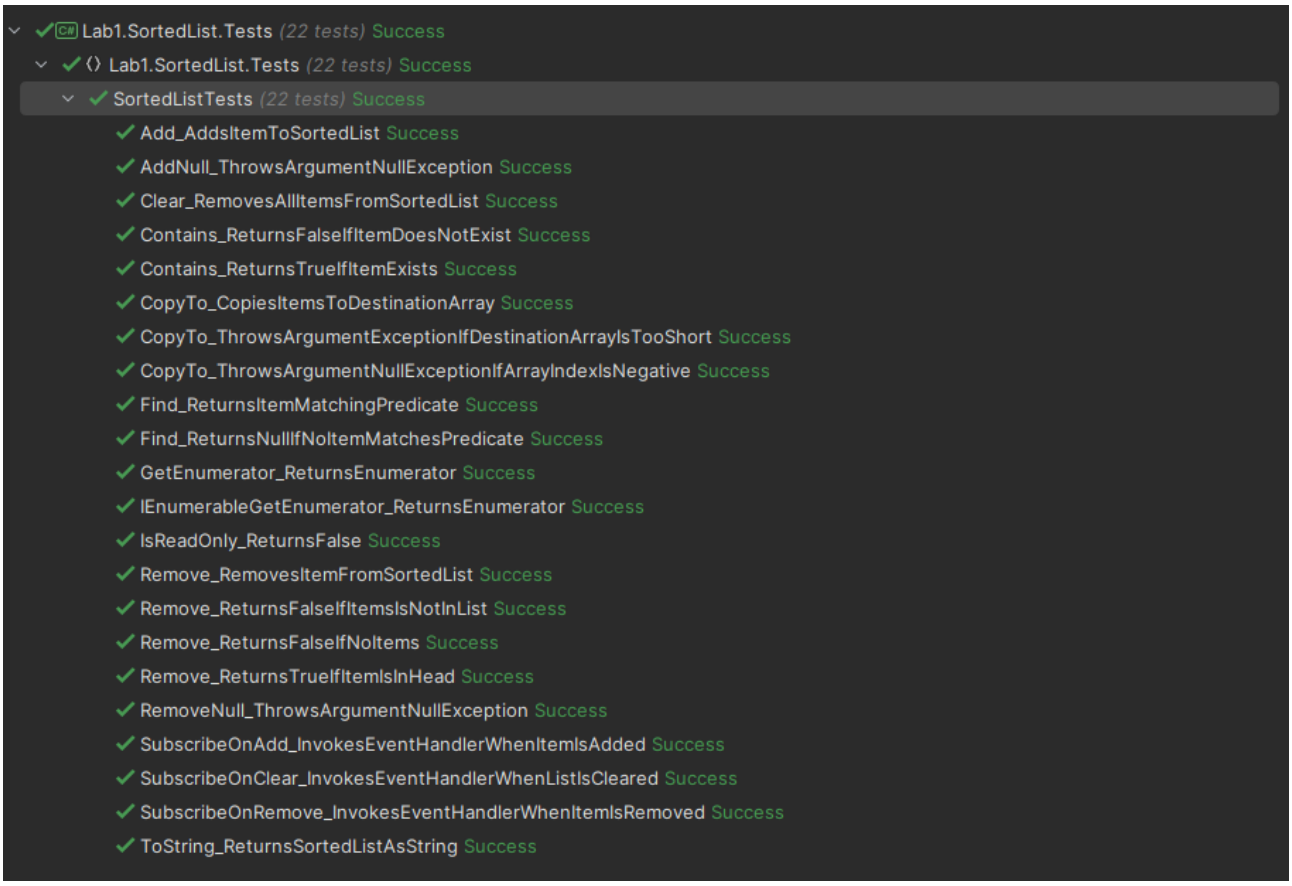
Мета лабораторної роботи – навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

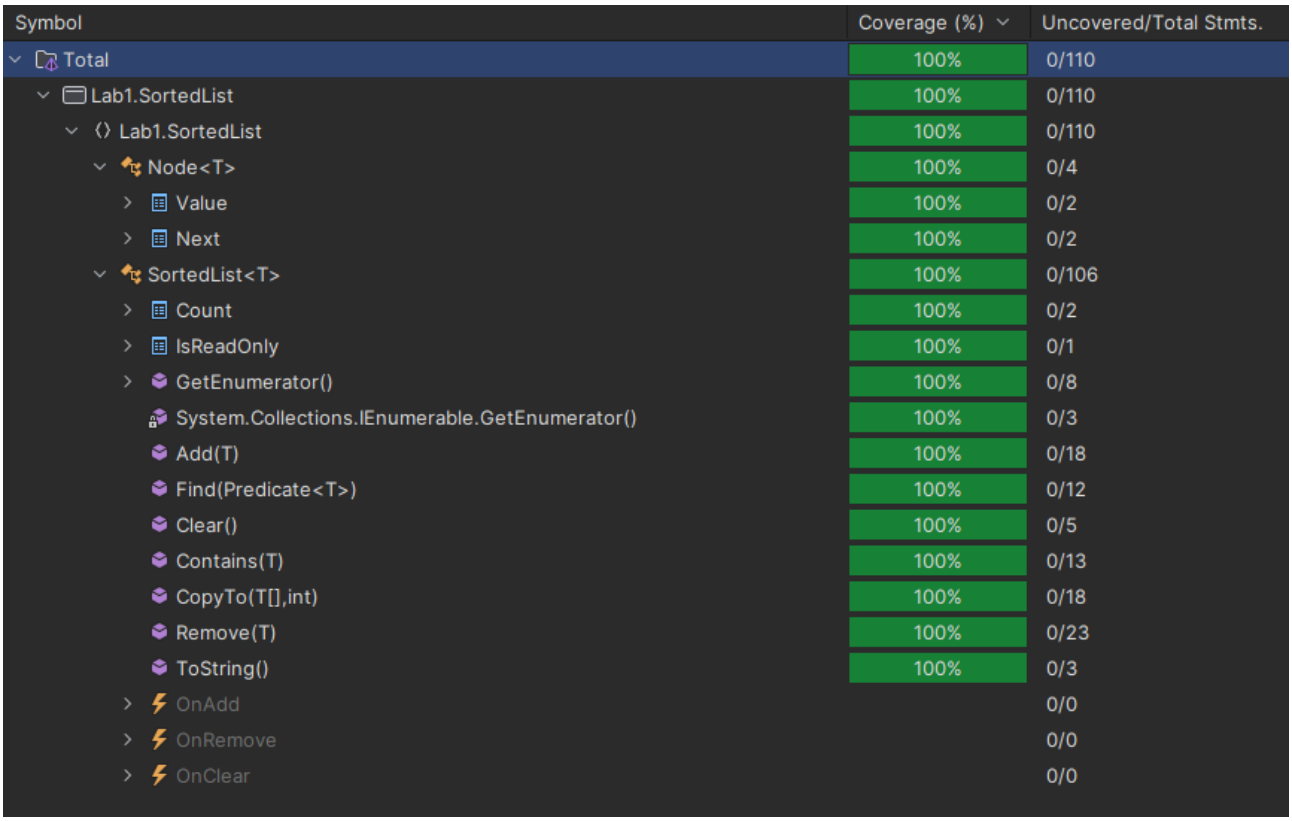
Завдання:
1. Додати до проекту власної узагальненої колекції (застосувати виконану лабораторну роботу No1) проект модульних тестів, використовуючи певний фреймворк (Nunit, Xunit, тощо).
2. Розробити модульні тести для функціоналу колекції.
3. Дослідити ступінь покриття модульними тестами вихідного коду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

Варіант:

7	Відсортований динамічний масив	Див. SortedList<T>	Збереження даних за допомогою динамічно зв'язаного списку або вектору

Результати:


Lab1.SortedList.Tests (22 tests) Success
Lab1.SortedList.Tests (22 tests) Success
SortedListTests (22 tests) Success
Add_AddsItemToSortedList Success
AddNull_ThrowsArgumentNullException Success
Clear_RemovesAllItemsFromSortedList Success
Contains_ReturnsFalseIfItemDoesNotExist Success
Contains_ReturnsTrueIfItemExists Success
CopyTo_CopiesItemsToDestinationArray Success
CopyTo_ThrowsArgumentExceptionIfDestinationArrayIsTooShort Success
CopyTo_ThrowsArgumentNullExceptionIfArrayIndexIsNegative Success
Find_ReturnsItemMatchingPredicate Success
Find_ReturnsNullIfNoItemMatchesPredicate Success
GetEnumerator_ReturnsEnumerator Success
IEnumerableGetEnumerator_ReturnsEnumerator Success
IsReadOnly_ReturnsFalse Success
Remove_RemovesItemFromSortedList Success
Remove_ReturnsFalseIfItemIsNotInList Success
Remove_ReturnsFalseIfNoItems Success
Remove_ReturnsTrueIfItemIsInHead Success
RemoveNull_ThrowsArgumentNullException Success
SubscribeOnAdd_InvokesEventHandlerWhenItemIsAdded Success
SubscribeOnClear_InvokesEventHandlerWhenListIsCleared Success
SubscribeOnRemove_InvokesEventHandlerWhenItemIsRemoved Success
ToString_ReturnsSortedListAsString Success


Symbol
Coverage (%)
Uncovered/Total Stmts.
Total 100% 0/110
Lab1.SortedList 100% 0/110
Lab1.SortedList 100% 0/110
Node<T> 100% 0/4
Value 100% 0/2
Next 100% 0/2
SortedList<T> 100% 0/106
Count 100% 0/2
IsReadOnly 100% 0/1
GetEnumerator() 100% 0/8
System.Collections.IEnumerable.GetEnumerator() 100% 0/3
Add(T) 100% 0/18
Find(Predicate<T>) 100% 0/12
Clear() 100% 0/5
Contains(T) 100% 0/13
CopyTo(T[],int) 100% 0/18
Remove(T) 100% 0/23
ToString() 100% 0/3
OnAdd 0/0
OnRemove 0/0
OnClear 0/0

Реалізація:

```
using System.Collections;

namespace Lab1.SortedList.Tests;
using FluentAssertions;

public class SortedListTests
{
    [Fact]
    public void Add_AddsItemToSortedList()
    {
        // Arrange
        var sortedList = new SortedList<int>();

        // Act
        sortedList.Add(3);
        sortedList.Add(1);
        sortedList.Add(2);

        // Assert
        sortedList.Should().BeInAscendingOrder();
        sortedList.Should().HaveCount(3);
        sortedList.Should().ContainInOrder(1, 2, 3);
    }

    [Fact]
    public void AddNull_ThrowsArgumentNullException()
    {
        // Arrange
        var sortedList = new SortedList<string>();

        // Act
        var addAction = () => sortedList.Add(null!);

        // Assert
        addAction.Should().Throw<ArgumentNullException>();
    }

    [Fact]
    public void Remove_RemovesItemFromSortedList()
    {
        // Arrange
        var sortedList = new SortedList<string>();
        sortedList.Add("a");
        sortedList.Add("b");
        sortedList.Add("c");
        sortedList.Add("d");

        // Act
        var result = sortedList.Remove("c");

        // Assert
        result.Should().BeTrue();
        sortedList.Should().HaveCount(3);
        sortedList.Should().ContainInOrder("a", "b", "d");
```

```
    }

    [Fact]
    public void Remove_ReturnsFalseIfNoItems()
    {
        // Arrange
        var sortedList = new SortedList<string>();

        // Act
        var result = sortedList.Remove("d");

        // Assert
        result.Should().BeFalse();
        sortedList.Should().HaveCount(0);
    }

    [Fact]
    public void Remove_ReturnsFalseIfItemsIsNotInList()
    {
        // Arrange
        var sortedList = new SortedList<string>();
        sortedList.Add("a");

        // Act
        var result = sortedList.Remove("d");

        // Assert
        result.Should().BeFalse();
        sortedList.Should().HaveCount(1);
    }

    [Fact]
    public void Remove_ReturnsTrueIfItemIsInHead()
    {
        // Arrange
        var sortedList = new SortedList<string>();
        sortedList.Add("a");

        // Act
        var result = sortedList.Remove("a");

        // Assert
        result.Should().BeTrue();
        sortedList.Should().HaveCount(0);
    }

    [Fact]
    public void RemoveNull_ThrowsArgumentNullException()
    {
        // Arrange
        var sortedList = new SortedList<string>();
        sortedList.Add("a");
        sortedList.Add("b");
        sortedList.Add("c");

        // Act
        var removeAction = () => sortedList.Remove(null!);
```

```
        // Assert
        removeAction.Should().Throw<ArgumentNullException>();
    }

    [Fact]
    public void Find_ReturnsItemMatchingPredicate()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        sortedList.Add(3);

        // Act
        var result = sortedList.Find(x => x == 2);

        // Assert
        result.Should().Be(2);
    }

    [Fact]
    public void Find_ReturnsNullIfNoItemMatchesPredicate()
    {
        // Arrange
        var sortedList = new SortedList<string>();
        sortedList.Add("a");
        sortedList.Add("b");
        sortedList.Add("c");

        // Act
        var result = sortedList.Find(x => x == "e");

        // Assert
        result.Should().BeNull();
    }

    [Fact]
    public void Clear_RemovesAllItemsFromSortedList()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        sortedList.Add(3);

        // Act
        sortedList.Clear();

        // Assert
        sortedList.Should().BeEmpty();
    }

    [Fact]
    public void Contains_ReturnsTrueIfItemExists()
    {
        // Arrange
```

```
        var sortedList = new SortedList<string>();
        sortedList.Add("apple");
        sortedList.Add("banana");
        sortedList.Add("cherry");

        // Act
        var contains = sortedList.Contains("banana");

        // Assert
        contains.Should().BeTrue();
    }

    [Fact]
    public void Contains_ReturnsFalseIfItemDoesNotExist()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(3);
        sortedList.Add(5);

        // Act
        var contains = sortedList.Contains(2);

        // Assert
        contains.Should().BeFalse();
    }

    [Fact]
    public void CopyTo_CopiesItemsToDestinationArray()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        sortedList.Add(3);
        var destinationArray = new int[3];

        // Act
        sortedList.CopyTo(destinationArray, 0);

        // Assert
        destinationArray.Should().ContainInOrder(1, 2, 3);
    }

    [Fact]
    public void CopyTo_ThrowsArgumentNullExceptionIfArrayIndexIsNegative()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        sortedList.Add(3);
        var destinationArray = new int[3];

        // Act
        var copyAction = () => sortedList.CopyTo(destinationArray, -1);
```

```
        // Assert
        copyAction.Should().Throw<ArgumentOutOfRangeException>();
    }

    [Fact]
    public void CopyTo_ThrowsArgumentExceptionIfDestinationArrayIsTooShort()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        var destinationArray = new int[1];

        // Act
        var copyAction = () => sortedList.CopyTo(destinationArray, 0);

        // Assert
        copyAction.Should().Throw<ArgumentException>();
    }

    [Fact]
    public void SubscribeOnAdd_InvokesEventHandlerWhenItemIsAdded()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        var eventWasInvoked = false;
        sortedList.OnAdd += (_, _) => eventWasInvoked = true;

        // Act
        sortedList.Add(1);

        // Assert
        eventWasInvoked.Should().BeTrue();
    }

    [Fact]
    public void SubscribeOnRemove_InvokesEventHandlerWhenItemIsRemoved()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        var eventWasInvoked = false;
        sortedList.OnRemove += (_, _) => eventWasInvoked = true;
        sortedList.Add(1);

        // Act
        sortedList.Remove(1);

        // Assert
        eventWasInvoked.Should().BeTrue();
    }

    [Fact]
    public void SubscribeOnClear_InvokesEventHandlerWhenListIsCleared()
    {
        // Arrange
        var sortedList = new SortedList<int>();
```

```
        var eventWasInvoked = false;
        sortedList.OnClear += (_, _) => eventWasInvoked = true;
        sortedList.Add(1);

        // Act
        sortedList.Clear();

        // Assert
        eventWasInvoked.Should().BeTrue();
    }

    [Fact]
    public void GetEnumerator_ReturnsEnumerator()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);

        // Act
        using var enumerator = sortedList.GetEnumerator();

        // Assert
        enumerator.Should().NotBeNull();
        enumerator.MoveNext().Should().BeTrue();
        enumerator.Current.Should().Be(1);
        enumerator.MoveNext().Should().BeTrue();
        enumerator.Current.Should().Be(2);
        enumerator.MoveNext().Should().BeFalse();
    }

    [Fact]
    public void IEnumerableGetEnumerator_ReturnsEnumerator()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);

        // Act
        var enumerator = ((IEnumerable) sortedList).GetEnumerator();

        // Assert
        enumerator.Should().NotBeNull();
        enumerator.MoveNext().Should().BeTrue();
        enumerator.Current.Should().Be(1);
        enumerator.MoveNext().Should().BeTrue();
        enumerator.Current.Should().Be(2);
        enumerator.MoveNext().Should().BeFalse();
    }

    [Fact]
    public void IsReadOnly_ReturnsFalse()
    {
        // Arrange
        var sortedList = new SortedList<int>();
```

```
        // Act
        var isReadOnly = sortedList.IsReadOnly;

        // Assert
        isReadOnly.Should().BeFalse();
    }

    [Fact]
    public void ToString_ReturnsSortedListAsString()
    {
        // Arrange
        var sortedList = new SortedList<int>();
        sortedList.Add(1);
        sortedList.Add(2);
        sortedList.Add(3);

        // Act
        var result = sortedList.ToString();

        // Assert
        result.Should().Be("1, 2, 3");
    }
}
```