МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

ДО ЛАБОРАТОРНОЇ РОБОТИ №2

з дисципліни

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

студента III курсу ФІОТ групи ІК-12

Басюка Валентина

Перевірив:
Бардін Владислав

Київ 2023

ЛАБОРАТОРНА РОБОТА №2

Модульне тестування. Ознайомлення з засобами та практиками модульного тестування

Мета: Навчитися створювати модульні тести для вихідного коду розроблювального програмного забезпечення.

Варіант №1

Завдання:

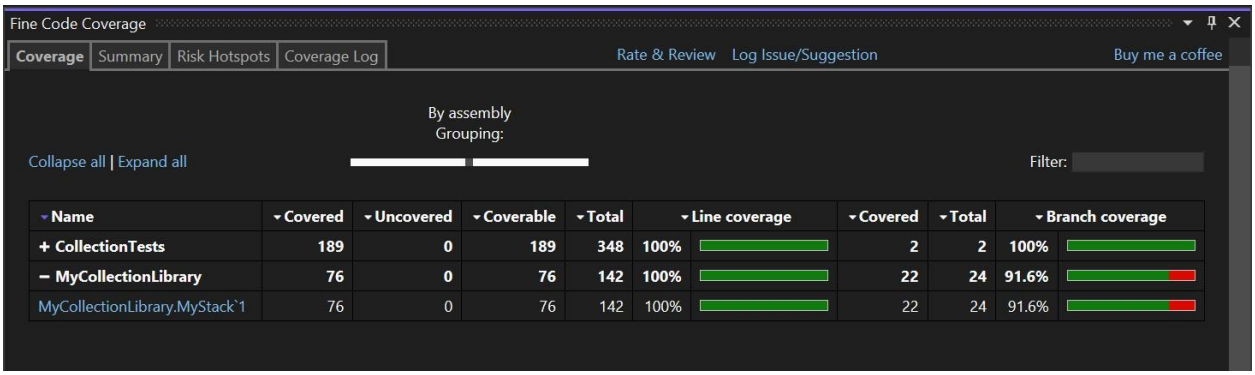
1. Додати до проекту власної узагальненої колекції (застосувати виконану лабораторну роботу No1) проект модульних тестів, використовуючи певний фреймворк (Nunit, Xunit, тощо).

2. Розробити модульні тести для функціоналу колекції.

3. Дослідити ступінь покриття модульними тестами вихідного коду колекції, використовуючи, наприклад, засіб AxoCover.

Варіант	Опис узагальненої колекції	Функціонал	Реалізація
1	Стек	Див. Stack<T>	Збереження даних за допомогою динамічно зв'язаного списку

Ступінь покриття модульними тестами вихідного коду:



Лістинг програмного коду

```
using System.Collections;
using System.Security.Cryptography.X509Certificates;
using MyCollectionLibrary;

namespace CollectionTests
{
    public class StackTests
    {
        public class PushTests
        {
            [Fact]
            public void Should_Add_Item_To_The_Top()
            {
                var stack = new MyStack<int>();

                stack.Push(1);
                stack.Push(2);
                stack.Push(3);

                Assert.True(stack.Contains(1));
                Assert.True(stack.Contains(2));
                Assert.Equal(3, stack.Peek());
            }

            [Fact]
            public void Should_Invoke_Add_Event()
            {
                var stack = new MyStack<int>();
                int pushedItem = 0;
                stack.OnItemAdded += (sender, item) => pushedItem = item;

                stack.Push(1);

                Assert.Equal(1, pushedItem);
            }
        }

        public class PopTests
        {
            [Fact]
            public void Should_Return_Top_Item_And_Remove_It()
            {
                var stack = new MyStack<int>();
                stack.Push(1);
                stack.Push(2);
                stack.Push(3);

                var poppedItem = stack.Pop();

                Assert.Equal(3, poppedItem);
                Assert.False(stack.Contains(3));
            }

            [Fact]
            public void Should_Invoke_Remove_Event()
            {
                var stack = new MyStack<int>();
                int removedItem = 0;
                stack.OnItemRemoved += (sender, item) => removedItem = item;

                stack.Push(1);
                stack.Push(2);
                stack.Push(3);
```

```
            stack.Pop();

            Assert.Equal(3, removedItem);
        }

        [Fact]
        public void Should_Throw_InvalidOperationException_On_Empty_Stack()
        {
            var stack = new MyStack<int>();

            Assert.Throws<InvalidOperationException>(() => stack.Pop());
        }
    }

    public class PeekTests
    {
        [Fact]
        public void Should_Return_Top_Item_Without_Removing()
        {
            var stack = new MyStack<int>();
            stack.Push(1);
            stack.Push(2);
            stack.Push(3);

            var peekedValue = stack.Peek();

            Assert.Equal(3, peekedValue);
            Assert.True(stack.Contains(3));
        }

        [Fact]
        public void Should_Throw_InvalidOperationException_On_Empty_Stack()
        {
            var stack = new MyStack<int>();

            Assert.Throws<InvalidOperationException>(() => stack.Peek());
        }
    }

    public class IsEmptyTests
    {
        [Fact]
        public void Should_Return_True_When_Stack_Empty()
        {
            var stack = new MyStack<int>();

            Assert.True(stack.IsEmpty());
        }

        [Fact]
        public void Should_Return_False_When_Stack_Not_Empty()
        {
            var stack = new MyStack<int>();

            stack.Push(1);
            stack.Push(2);

            Assert.False(stack.IsEmpty());
        }
    }
```

```
public class ClearTests
{
    [Fact]
    public void Should_Clear_Stack()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        stack.Clear();

        Assert.Empty(stack);
    }

    [Fact]
    public void Should_Invoke_Cleared_Event()
    {
        var stack = new MyStack<int>();
        bool clearedEventInvoked = false;
        stack.OnClear += (sender, item) => clearedEventInvoked = true;

        stack.Push(1);
        stack.Push(2);
        stack.Clear();

        Assert.True(clearedEventInvoked);
    }
}

public class ContainsTests
{
    [Fact]
    public void Should_Return_True_When_Item_Exist()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        Assert.True(stack.Contains(2));
    }

    [Fact]
    public void Should_Return_False_When_Item_Not_Exist()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        Assert.False(stack.Contains(4));
    }

    [Fact]
    public void Should_Return_False_When_Stack_Empty()
    {
        var stack = new MyStack<int>();

        Assert.False(stack.Contains(1));
    }
}
```

```
public class ToStringTests
{
    [Fact]
    public void Should_Return_String_Stack_Representation()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);

        var stackToString = stack.ToString();

        Type result = stackToString.GetType();

        string str = "";
        Type expected = str.GetType();

        Assert.Equal(expected, result);
    }

    [Fact]
    public void Should_Return_Correct_String_Items_Representation()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        var stackToString = stack.ToString();

        Assert.Equal("3\r\n2\r\n1\r\n", stackToString);
    }
}

public class GetEnumeratorTests
{
    [Fact]
    public void Should_Return_Generic_Enumerator()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        IEnumerator<int> enumerator = stack.GetEnumerator();

        Assert.NotNull(enumerator);
    }

    [Fact]
    public void Should_Return_Non_Generic_Enumerator()
    {
        var stack = new MyStack<int>();
        stack.Push(1);
        stack.Push(2);
        stack.Push(3);

        IEnumerator enumerator = stack.GetEnumerator();

        Assert.NotNull(enumerator);
    }

    [Fact]
    public void Should_Traverse_Stack_Items()
    {
        var stack = new MyStack<int>();
```

```
            stack.Push(1);
            stack.Push(2);
            stack.Push(3);

            List<int> items = new List<int>();
            foreach (int item in stack)
            {
                items.Add(item);
            }

            Assert.Equal(new List<int> { 3, 2, 1 }, items);
        }

    }

    public class MyEnumeratorTests
    {
        [Fact]
        public void Should_Return_Current_Element()
        {
            var stack = new MyStack<int>();
            stack.Push(1);
            stack.Push(2);
            var enumerator = stack.GetEnumerator();
            enumerator.MoveNext();
            var current = enumerator.Current;

            Assert.Equal(2, current);
        }

        [Fact]
        public void Should_Return_Current_Element_Non_Generic_Enumerator()
        {
            var stack = new MyStack<int>();
            stack.Push(1);
            stack.Push(2);
            IEnumerator enumerator = stack.GetEnumerator();
            enumerator.MoveNext();
            var current = enumerator.Current;

            Assert.Equal(2, current);
        }

        [Fact]
        public void Current_Should_Return_Exeption()
        {
            var stack = new MyStack<int>();

            IEnumerator<int> enumerator = stack.GetEnumerator();

            Assert.Throws<InvalidOperationException>(() => enumerator.Current);
        }

        [Fact]
        public void Should_Move_To_Next_Element()
        {
            var stack = new MyStack<int>();

            stack.Push(1);
            stack.Push(2);

            var enumerator = stack.GetEnumerator();

            var result1 = enumerator.MoveNext();
            var result2 = enumerator.MoveNext();
```

```
            var result3 = enumerator.MoveNext();

            Assert.True(result1);
            Assert.True(result2);
            Assert.False(result3);
        }

        [Fact]
        public void Should_Reset_Enumerator()
        {
            var stack = new MyStack<int>();

            stack.Push(1);
            stack.Push(2);

            var enumerator = stack.GetEnumerator();

            enumerator.MoveNext();

            var current = enumerator.Current;

            enumerator.Reset();

            enumerator.MoveNext();

            Assert.Equal(current, enumerator.Current);
        }
    }
}
}
```