**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 2 з дисципліни

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

«Модульне тестування. Ознайомлення з засобами та практиками

модульного тестування»

**Виконав**

(шифр, прізвище, ім'я, по батькові)

*ІС-13,Харчук А.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Бардін В.*

Київ 2023

Варіант 5

Завдання



Код тестів

using System.Collections;

using System.Collections.Generic;

namespace CollectionRealisation;

*public* class StackTests

{

*public* class PropertiesTests

    {

        [Fact]

*public* void Constructor\_WithNegativeCapacity\_ThrowsArgumentOutOfRangeException()

        {

*// Arrange*

            int capacity = -1;

*// Act and Assert*

            Assert.Throws<ArgumentOutOfRangeException>(() => new MyStack<int>(capacity));

        }

        [Fact]

*public* void Count\_Property\_Should\_Return\_Correct\_Value()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            Assert.Equal(3, stack.Count);

        }

        [Fact]

*public* void IsReadOnly\_Property\_Should\_Always\_Return\_False()

        {

            var stack = new MyStack<int>();

            Assert.Equal(false, stack.IsReadOnly);

        }

        [Fact]

*public* void IsSynchronized\_Property\_Should\_Always\_Return\_False()

        {

            var stack = new MyStack<int>();

            Assert.Equal(false, stack.IsSynchronized);

        }

        [Fact]

*public* void SyncRoot\_Property\_Should\_Return\_This\_Instance()

        {

            var stack = new MyStack<int>();

            Assert.Equal(stack, stack.SyncRoot);

        }

        [Fact]

*public* void ToString\_Should\_Return\_Correct\_String\_Representation()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            string result = stack.ToString();

            Assert.Equal("Count = 2", result);

        }

    }

*public* class ClearTests

    {

        [Fact]

*public* void Clear\_Should\_Clear\_Stack\_And\_Invoke\_Cleared\_Event()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            stack.Clear();

            Assert.Empty(stack);

        }

        [Fact]

*public* void Clear\_Should\_Invoke\_Cleared\_Event()

        {

            var stack = new MyStack<int>();

            bool clearedEventInvoked = false;

            stack.Cleared += () => clearedEventInvoked = true;

            stack.Push(1);

            stack.Clear();

            Assert.True(clearedEventInvoked);

        }

    }

*public* class ContainsTests

    {

        [Fact]

*public* void Contains\_Should\_Return\_True\_When\_Item\_Exists()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

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

        }

        [Fact]

*public* void Contains\_Should\_Return\_False\_When\_Item\_Does\_Not\_Exist()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

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

        }

        [Fact]

*public* void Contains\_Should\_Return\_False\_For\_Empty\_Stack()

        {

            var stack = new MyStack<int>();

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

        }

    }

*public* class GetEnumeratorTests

    {

        [Fact]

*public* void GetEnumerator\_Should\_Return\_Enumerator\_For\_Generic\_Enumerable()

        {

            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 GetEnumerator\_Should\_Return\_Non\_Generic\_Enumerator()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

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

            Assert.NotNull(enumerator);

        }

        [Fact]

*public* void GetEnumerator\_Should\_Traverse\_Stack\_Elements()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            List<int> elements = new List<int>();

            foreach (int item in stack)

            {

                elements.Add(item);

            }

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

        }

    }

*public* class PeekTests

    {

        [Fact]

*public* void Peek\_Should\_Return\_Top\_Element\_Without\_Removing\_It()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            var peekedValue = stack.Peek();

            Assert.Equal(3, peekedValue);

            Assert.Equal(3, stack.Count);

        }

        [Fact]

*public* void Peek\_Should\_Throw\_InvalidOperationException\_On\_Empty\_Stack()

        {

            var stack = new MyStack<int>();

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

        }

    }

*public* class PopTests

    {

        [Fact]

*public* void Pop\_Should\_Return\_Top\_Element\_And\_Remove\_It()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            var poppedValue = stack.Pop();

            Assert.Equal(3, poppedValue);

            Assert.Equal(2, stack.Count);

        }

        [Fact]

*public* void Pop\_Should\_Throw\_InvalidOperationException\_On\_Empty\_Stack()

        {

            var stack = new MyStack<int>();

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

        }

    }

*public* class PushTests

    {

        [Fact]

*public* void Push\_Should\_Add\_Item\_To\_The\_Top\_Of\_The\_Stack()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            Assert.Equal(2, stack.Count);

            Assert.Equal(2, stack.Peek());

        }

        [Fact]

*public* void Push\_Should\_Invoke\_Pushed\_Event()

        {

            var stack = new MyStack<int>();

            int pushedValue = 0;

            stack.Pushed += value => pushedValue = value;

            stack.Push(3);

            Assert.Equal(3, pushedValue);

        }

        [Fact]

*public* void Push\_Should\_Add\_Capacity()

        {

            var stack = new MyStack<int>();

            var defaultCapacity = MyStack<int>.DefaultCapacity;

            for (int i = 0; i < defaultCapacity; i++)

            {

                stack.Push(default);

            }

            stack.Push(1);

            Assert.Equal(stack.Count, defaultCapacity+1);

        }

    }

*public* class CopyToTests

    {

        [Fact]

*public* void CopyTo\_Should\_Copy\_Stack\_Elements\_To\_Array()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            stack.Push(3);

            int[] array = new int[5];

            stack.CopyTo(array, 1);

            Assert.Equal(new int[] { 0, 3, 2, 1, 0 }, array);

        }

        [Fact]

*public* void CopyTo\_Should\_Throw\_ArgumentNullException\_If\_Array\_Is\_Null()

        {

            var stack = new MyStack<int>();

            int[] array = null!;

            Assert.Throws<ArgumentNullException>(() => stack.CopyTo(array!, 0));

        }

        [Fact]

*public* void CopyTo\_Should\_Throw\_InvalidDataException\_If\_Array\_Rank\_Is\_Not\_1()

        {

            var stack = new MyStack<int>();

            int[,] array = new int[2, 2];

            Assert.Throws<InvalidDataException>(() => stack.CopyTo(array, 0));

        }

        [Fact]

*public* void CopyTo\_Should\_Throw\_ArgumentException\_If\_Array\_Type\_Mismatch()

        {

            var stack = new MyStack<int>();

            double[] array = new double[5];

            Assert.Throws<ArgumentException>(() => stack.CopyTo(array, 0));

        }

        [Fact]

*public* void CopyTo\_Should\_Copy\_Empty\_Stack\_To\_Array()

        {

            var stack = new MyStack<int>();

            int[] array = new int[5];

            stack.CopyTo(array, 1);

            Assert.Equal(new int[] { 0, 0, 0, 0, 0 }, array);

        }

    }

*public* class IEnumeratorTests

    {

        [Fact]

*public* void Current\_Property\_Returns\_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 IEnumerator\_Current\_Property\_Returns\_Current\_Element()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            var enumerator = stack.GetEnumerator();

            enumerator.MoveNext();

            var current = (enumerator as IEnumerator)?.Current;

            Assert.Equal(2, current);

        }

        [Fact]

*public* void MoveNext\_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 Reset\_Should\_Reset\_Enumerator()

        {

            var stack = new MyStack<int>();

            stack.Push(1);

            stack.Push(2);

            var enumerator = stack.GetEnumerator();

            enumerator.MoveNext();

            var firstCurrent = enumerator.Current;

            enumerator.Reset();

            enumerator.MoveNext();

            var secondCurrent = enumerator.Current;

            Assert.Equal(firstCurrent, secondCurrent);

        }

    }

}