Національний технічний університет України
«КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра Інформаційних Систем та Технологій

Комп’ютерний практикум №2

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

на тему

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

Виконав:
студент гр. ІС-11
Побережний Олександр

Київ 2023

Лістинг програмного коду:

CustomLinkedList.cs

```
using System.Collections;

namespace DataLayer
{
    public class CustomLinkedList<T> : ICollection<T>
    {
        public delegate void OnActionEventHandler(object sender);
        public event OnActionEventHandler? OnAdding;
        public event OnActionEventHandler? OnRemoving;
        public event OnActionEventHandler? OnCleared;
        public event OnActionEventHandler? OnCopied;
        public event OnActionEventHandler? OnEndPlaced;
        public event OnActionEventHandler? OnBeginPlaced;

        public Node<T>? First { get; private set;}
        public Node<T>? Last { get; private set;}
        public int Count { get; private set;}
        public bool IsReadOnly => false;

        public CustomLinkedList()
        {
            this.First = null;
            this.Last = null;
        }

        public void AddFirst(Node<T> node)
        {
            if(IsEmpty())
            {
                this.First = node;
                this.Last = node;
            }
            else
            {
                this.First!.Previous = node;
                node.Next = this.First;
                this.First = node;
            }
            Count++;

            OnBeginPlaced?.Invoke(this);
        }

        public void AddLast(Node<T> node)
        {
            if (IsEmpty())
            {
                this.First = node;
                this.Last = node;
            }
            else
            {
                this.Last!.Next = node;
                node.Previous = this.Last;
                this.Last = node;
            }


            Count++;
            OnEndPlaced?.Invoke(this);
```

```
}

public void Clear()
{
    First = null;
    Last = null;
    Count = 0;

    OnCleared?.Invoke(this);
}

public void AddBefore(Node<T> newNode, Node<T> oldNode)
{
    if (IsEmpty() || this.First == oldNode)
    {
        this.AddFirst(newNode);
        return;
    }

    if (!this.Contains(oldNode))
    {
        throw new NullReferenceException("Such node does not exist");
    }

    Node<T>? prevNode = oldNode.Previous;

    newNode.Previous = prevNode;
    newNode.Next = oldNode;
    oldNode.Previous = newNode;
    prevNode!.Next = newNode;

    Count++;

    OnAdding?.Invoke(this);
}

public void AddAfter(Node<T> newNode, Node<T> oldNode)
{
    if(IsEmpty() || (this.First == oldNode && this.Last == oldNode))
    {
        AddLast(newNode);
        return;
    }

    if(this.Contains(oldNode) == false)
    {
        throw new NullReferenceException("Such a node does not exists");
    }

    Node<T>? nextNode = oldNode.Next;

    newNode.Previous = oldNode;
    newNode.Next = nextNode;
    nextNode!.Previous = newNode;
    oldNode.Next = newNode;

    Count++;

    OnAdding?.Invoke(this);
}

public void RemoveFirst()
{
    if(IsEmpty())
    {
        throw new Exception("There is no item to remove");
    }
```

```
        if(Count == 1)
        {
            Clear();
            return;
        }

        this.First = this.First!.Next;
        this.First!.Previous = null;
        Count--;

        OnRemoving?.Invoke(this);
    }

    public void RemoveLast()
    {
        if (this.Count == 0)
        {
            throw new Exception("There is nothing to remove");
        }

        this.Last = this.Last!.Previous;
        this.Last!.Next = null;
        Count--;

        OnRemoving?.Invoke(this);
    }

    public bool Remove(T node)
    {
        if(IsEmpty())
        {
            throw new Exception("There is nothing to remove");
        }
        else if(First!.Equals(node))
        {
            RemoveFirst();
            return true;
        }
        else
        {
            Node<T>? prevNode = First;
            Node<T>? currNode = prevNode.Next;

            while(currNode != null && !currNode.Equals(node))
            {
                prevNode = currNode;
                currNode = prevNode.Next;
            }

            if (currNode != null)
            {
                prevNode.Next = currNode.Next;
                currNode.Next = currNode.Previous;
            }
            Count--;
            OnRemoving?.Invoke(this);
            return true;

        }
    }

    public void Add(T item)
    {
        AddLast(new Node<T>(item));

        OnAdding?.Invoke(this);
    }
```

```
        public bool Contains(T item)
        {
            Node<T> node = First!;
            while(!node.Equals(item) && node.Next != null)
            {
                node = node.Next;
            }
            if (node.Equals(item)) return true;
            else return false;

        }

        private bool Contains(Node<T> item)
        {
            Node<T> node = First!;
            while (!node.Equals(item) && node.Next != null)
            {
                node = node.Next;
            }
            if (node.Equals(item)) return true;
            else return false;
        }

        public void CopyTo(T[] array, int arrayIndex)
        {
            if (arrayIndex < 0 || arrayIndex > array.Length)
            {
                throw new ArgumentOutOfRangeException("Index was out of range");
            }

            if (array.Length - arrayIndex < Count)
            {
                throw new InvalidOperationException("Cannot copy to array: not enough
space");
            }

            Node<T>? node = First;

            while(node != null)
            {
                array[arrayIndex++] = node.Data;
                node = node.Next;
            }

            OnCopied?.Invoke(this);
        }

        public Node<T> Find(T item)
        {
            Node<T> node = First!;
            EqualityComparer<T> comparer = EqualityComparer<T>.Default;
            if(node != null)
            {
                if(item != null )
                {
                    while (node != null)
                    {
                        if (comparer.Equals(node.Data, item))
                        {
                            return node;
                        }
                        node = node.Next!;
                    }
                }
                else
                {
                    throw new InvalidOperationException("Item is not specified");
                }
```

```
            }
            return null;
        }


        public IEnumerator<T> GetEnumerator()
        {
            Node<T> node = First!;

            while(node != null)
            {
                yield return node.Data;
                node = node.Next;
            }
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            Node<T> node = First!;

            while (node != null)
            {
                yield return node.Data;
                node = node.Next;
            }
        }

        private bool IsEmpty()
        {
            return this.First == null && this.Last == null && this.Count.Equals(0);
        }
    }
                                            }
```

CustomLinkedListTests.cs

```
namespace CustomLinkedListTesting;
using DataLayer;
using System.Diagnostics.Contracts;

[TestFixture]
public class Tests
{
    private CustomLinkedList<string> _list;
    [SetUp]
    public void Setup()
    {
        _list = new CustomLinkedList<string>();
    }

    [Test]
    public void AddFirst_NewNode_EmptyList_ResultListWithOneNode()
    {
        var node = new Node<string>("Node");

        _list.AddFirst(node);

        Assert.Multiple(() =>
        {
            Assert.That(_list.First, Is.EqualTo(node));
            Assert.That(_list, Has.Count.EqualTo(1));
        });
    }

    [Test]
    public void AddFirst_NewNode_ListWithOneNode_ResultFirstNodeIsAddedNode()
```

```
{
    var node = new Node<string>("oldNode");
    _list.AddFirst(node);

    var node1 = new Node<string>("newNode");
    _list.AddFirst(node1);

    Assert.Multiple(() =>
    {
        Assert.That(_list.First, Is.EqualTo(node1));
        Assert.That(_list, Has.Count.EqualTo(2));
    });
}

[Test]
public void AddLast_NewNode_EmptyList_ResultListWithOneNodeOnLastPlace()
{
    var node = new Node<string>("NewNode");

    _list.AddLast(node);

    Assert.Multiple(() =>
    {
        Assert.That(_list.Last, Is.EqualTo(node));
        Assert.That(_list.Last, Is.SameAs(node));
        Assert.That(_list, Has.Count.EqualTo(1));
    });
}

[Test]
public void AddLast_NewNode_ListWithOneNode_ResultLastNodeIsNewNode()
{
    var node = new Node<string>("oldNode");
    _list.AddLast(node);

    Node<string> node1 = new("NewNode");
    _list.AddLast(node1);

    Assert.Multiple(() =>
    {
        Assert.That(_list.Last, Is.EqualTo(node1));
        Assert.That(_list, Has.Count.EqualTo(2));
    });
}

[Test]
public void Clear_ListWith5Elements_ResultEmptyList()
{
    for(int i = 0; i < 5; i++)
    {
        _list.AddLast(new Node<string>($"node{i}"));
    }

    _list.Clear();

    Assert.Multiple(() =>
    {
        Assert.That(_list, Has.Count.EqualTo(0));
        Assert.True(_list.First == null && _list.Last == null);
    });

}

[Test]
public void AddBefore_EmptyList_ResultListWithOneElementOnFirstPlace()
{
    Node<string> node = new("newNode");
```

```
        _list.AddBefore(node, null);

        Assert.Multiple(() =>
        {
            Assert.That(_list.First, Is.EqualTo(node));
            Assert.That(_list.First, Is.SameAs(node));
            Assert.That(_list, Has.Count.EqualTo(1));
        });
    }


    [Test]
    public void AddBefore_ListWith1Node_Result2ElementsWithNewNodeOnTheFirstPlace()
    {
        Node<string> node = new("OldNode");
        _list.AddLast(node);

        Node<string> node1 = new("NewNode");

        _list.AddBefore(node1, node);


        Assert.Multiple(() =>
        {
            Assert.That(_list.First, Is.SameAs(node1));
            Assert.That(_list.First, Is.EqualTo(node1));
            Assert.That(_list, Has.Count.EqualTo(2));
        });
    }

    [Test]
    public void
AddBefore_ListWith1NodeButOldNodeIsNotFromList_ResultNullRefException()
    {
        Node<string> node = new("oldNode");

        Node<string> fakeOldNode = new("OldNode");

        _list.AddLast(node);

        Node<string> newNode = new("NewNode");

        Assert.Throws<NullReferenceException>(() => _list.AddBefore(newNode,
fakeOldNode));
    }

    [Test]
    public void AddBefore_ListWith3Elements_ResultAddedNodeBeforeLastElement()
    {
        Node<string> node1 = new("Node1");
        Node<string> node2 = new("Node2");
        Node<string> node3 = new("Node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);

        Node<string> node4 = new("Node4");

        _list.AddBefore(node4, node3);

        Assert.Multiple(() =>
        {
            Assert.That(_list, Has.Count.EqualTo(4));
            Assert.That(node3.Previous, Is.EqualTo(node4));
            Assert.That(node3.Previous, Is.SameAs(node4));
            Assert.That(_list.Last, Is.EqualTo(node3));
            Assert.That(_list.Last, Is.SameAs(node3));
```

```
        });

    }

    [Test]
    public void AddAfter_EmptyList_ResultListWith1ElementOnFirstPlace()
    {
        Node<string> node1 = new("Node");

        _list.AddAfter(node1, null);

        Assert.Multiple(() =>
        {
            Assert.That(_list, Has.Count.EqualTo(1));
            Assert.That(_list.First, Is.EqualTo(node1));
            Assert.That(_list.First, Is.SameAs(node1));
        });
    }

    [Test]
    public void
AddAfter_ListWith1Element_ResultListWith2ElementsWithNewNodeOnTheLastPlace()
    {
        Node<string> node1 = new("node1");

        _list.AddLast(node1);

        Node<string> node2 = new("node2");

        _list.AddAfter(node2, node1);

        Assert.Multiple(() =>
        {
            Assert.That(_list, Has.Count.EqualTo(2));
            Assert.That(_list.Last, Is.EqualTo(node2));
            Assert.That(_list.Last, Is.SameAs(node2));
            Assert.That(_list.First, Is.EqualTo(node1));
            Assert.That(_list.First, Is.SameAs(node1));
        });

    }

    [Test]
    public void
AddAfter_ListWith1NodeButOldNodeIsNotFromList_ResultThrowNullrefException()
    {
        Node<string> node1 = new("oldNode");
        Node<string> fakeNode = new("oldNode");
        _list.AddLast(node1);

        Node<string> node2 = new("newNode");

        Assert.Throws<NullReferenceException>(() => _list.AddAfter(node2, fakeNode));
    }

    [Test]
    public void AddAfter_ListWith3Elements_ResultAddedElementAfterFirst()
    {
        Node<string> node1 = new("Node1");
        Node<string> node2 = new("Node2");
        Node<string> node3 = new("Node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);

        Node<string> node4 = new("Node4");
```

```
    _list.AddAfter(node4, node1);

    Assert.Multiple(() =>
    {
        Assert.That(_list, Has.Count.EqualTo(4));
        Assert.That(node1.Next, Is.EqualTo(node4));
        Assert.That(node1.Next, Is.SameAs(node4));
        Assert.That(_list.First, Is.EqualTo(node1));
        Assert.That(_list.First, Is.SameAs(node1));
    });
}

[Test]
public void RemoveFirst_EmptyList_ResultThrowException()
{
    Assert.Throws<Exception>(() => _list.RemoveFirst());
}

[Test]
public void RemoveFirst_ListWith2Elements_ResultListWith1ElementOnTheFirstPlace()
{
    Node<string> node1 = new("Node1");
    Node<string> node2 = new("Node2");

    _list.AddLast(node1);
    _list.AddLast(node2);

    _list.RemoveFirst();

    Assert.Multiple(() =>
    {
        Assert.That(_list, Has.Count.EqualTo(1));
        Assert.That(_list.First, Is.EqualTo(node2));
        Assert.That(_list.First, Is.SameAs(node2));
    });
}

[Test]
public void RemoveLast_EmptyList_ResultThrowException()
{
    Assert.Throws<Exception>(() => _list.RemoveLast());
}

[Test]
public void RemoveLast_ListWith2Elements_ResultListWith1ElementOnTheFirstPlace()
{
    Node<string> node1 = new("Node1");
    Node<string> node2 = new("Node2");

    _list.AddLast(node1);
    _list.AddLast(node2);

    _list.RemoveLast();

    Assert.Multiple(() =>
    {
        Assert.That(_list, Has.Count.EqualTo(1));
        Assert.That(_list.First, Is.EqualTo(node1));
        Assert.That(_list.First, Is.SameAs(node1));
    });
}

[Test]
public void Remove_EmptyList_ResultThrowException()
{
    Assert.Throws<Exception>( () => _list.Remove("node"));
}
```

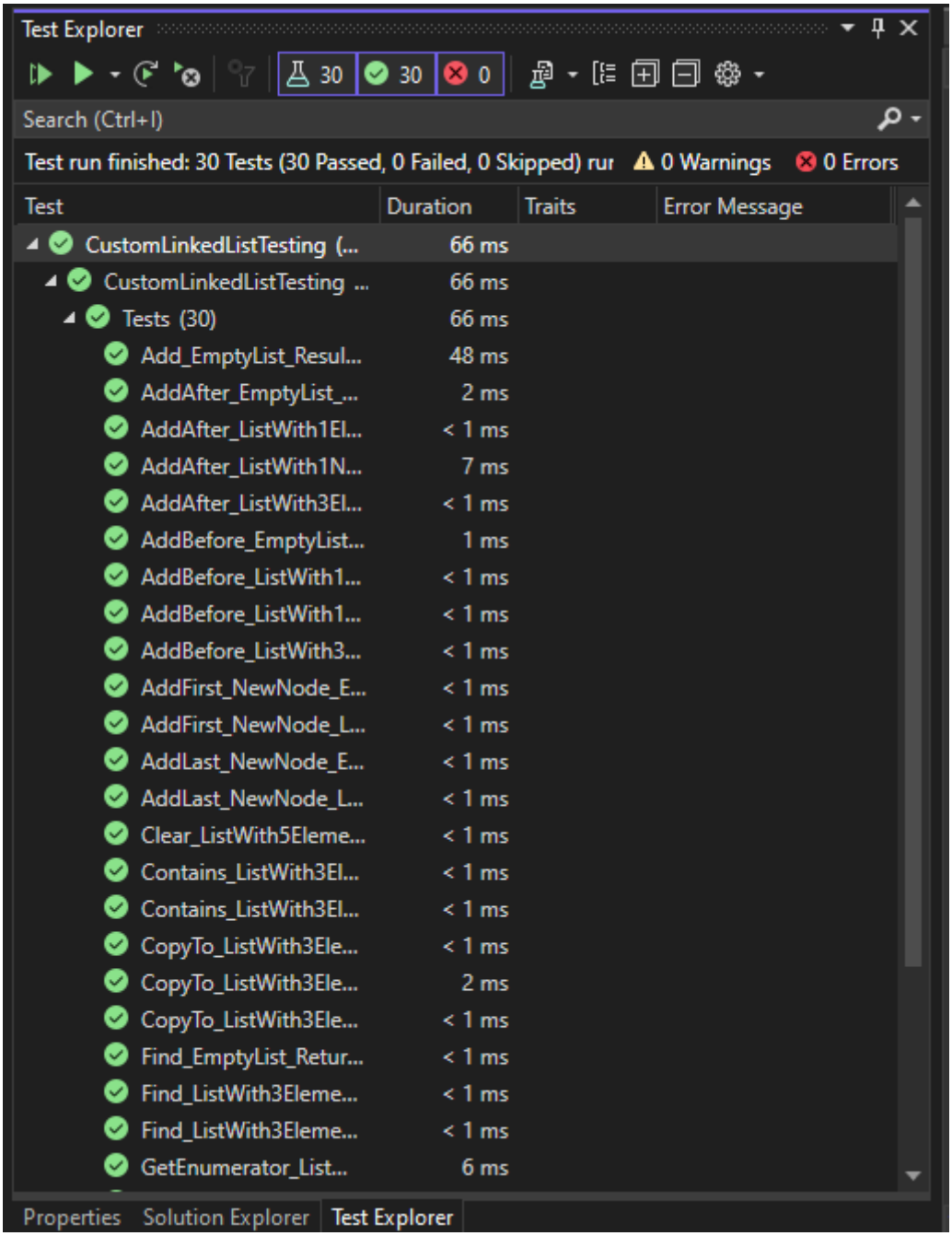
```
[Test]
public void Remove_ListWith1ElementOnFirstPlace_ResultEmptyListReturnedTrue()
{
    Node<string> node = new("Node");

    _list.AddFirst(node);

    var rem = _list.Remove("Node");

    Assert.Multiple(() =>
    {
        Assert.True(rem);
        Assert.That(_list, Has.Count.EqualTo(0));
        Assert.That(_list.First, Is.EqualTo(null));
        Assert.That(_list.Last, Is.EqualTo(null));
    });
}

[Test]
public void Remove_ListWith3Elements_ResultListWith2ElementsReturnedTrue()
{
    Node<string> node1 = new("node1");
    Node<string> node2 = new("node2");
    Node<string> node3 = new("node3");

    _list.AddLast(node1);
    _list.AddLast(node2);
    _list.AddLast(node3);

    var res = _list.Remove("node3");

    Assert.Multiple(() =>
    {
        Assert.That(res, Is.True);
        Assert.That(_list, Has.Count.EqualTo(2));

    });

}

[Test]
public void Add_EmptyList_ResultListWith1ElementOnTheFirstPlace()
{
    _list.Add("Node1");
    Assert.Multiple(() =>
    {
        Assert.That(_list, Has.Count.EqualTo(1));
        Assert.That(_list.First.Data, Is.EqualTo("Node1"));
    });
}

[Test]
public void Contains_ListWith3Elements_ReturnedTrue()
{
    Node<string> node1 = new("node1");
    Node<string> node2 = new("node2");
    Node<string> node3 = new("node3");

    _list.AddLast(node1);
    _list.AddLast(node2);
    _list.AddLast(node3);

    var res = _list.Contains("node2");

    Assert.That(res, Is.True);
}

[Test]
```

```
    public void Contains_ListWith3Elements_ReturnedFalse()
    {
        Node<string> node1 = new("node1");
        Node<string> node2 = new("node2");
        Node<string> node3 = new("node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);

        var res = _list.Contains("node11");

        Assert.That(res, Is.False);
    }

    [Test]
    public void
CopyTo_ListWith3ElementsAndWrongIndexInParams_ThrowArgumentOutOfrangeExcep()
    {
        string[] arr = new string[3];

        Node<string> node1 = new("node1");
        Node<string> node2 = new("node2");
        Node<string> node3 = new("node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);

        TestDelegate res = () => _list.CopyTo(arr, -1);

        Assert.Throws<ArgumentOutOfRangeException>(res);
    }

    [Test]
    public void
CopyTo_ListWith3ElementsAndIndexParamDoesNotFitToArray_ThrowInvalidOperationException(
)
    {
        string[] arr = new string[3];

        Node<string> node1 = new("node1");
        Node<string> node2 = new("node2");
        Node<string> node3 = new("node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);

        TestDelegate res = () => _list.CopyTo(arr, 2);

        Assert.Throws<InvalidOperationException>(res);
    }

    [Test]
    public void
CopyTo_ListWith3ElementsAndValidIndexParameter_ResultArrayWith3Elements()
    {
        string[] arr = new string[3];

        Node<string> node1 = new("node1");
        Node<string> node2 = new("node2");
        Node<string> node3 = new("node3");

        _list.AddLast(node1);
        _list.AddLast(node2);
        _list.AddLast(node3);
```

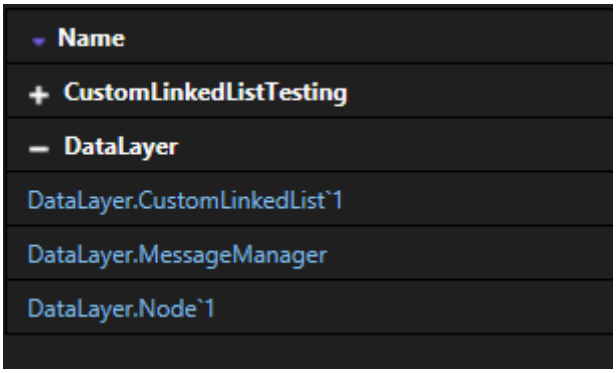
```
    _list.CopyTo(arr, 0);

    Assert.Multiple(() =>
    {
        Assert.That(arr, Has.Length.EqualTo(3));
        Assert.That(arr, Is.All.Not.Null);
    });
}

[Test]
public void Find_EmptyList_ReturnedNull()
{
    var node = _list.Find("Node");

    Assert.That(node, Is.Null);
}

[Test]
public void Find_ListWith3ElementsAndItemIsNotNull_ReturnedNode()
{
    Node<string> node1 = new("node1");
    Node<string> node2 = new("node2");
    Node<string> node3 = new("node3");

    _list.AddLast(node1);
    _list.AddLast(node2);
    _list.AddLast(node3);

    var node = _list.Find("node2");

    Assert.Multiple(() =>
    {
        Assert.That(node, Is.Not.Null);
        Assert.That(node, Is.EqualTo(node2));
        Assert.That(node, Is.SameAs(node2));
    });
}

[Test]
public void Find_ListWith3ElementsAndItemIsNull_ReturnedNull()
{
    Node<string> node1 = new("node1");
    Node<string> node2 = new("node2");
    Node<string> node3 = new("node3");

    _list.AddLast(node1);
    _list.AddLast(node2);
    _list.AddLast(node3);

    TestDelegate res = () => _list.Find(default);

    Assert.Throws<InvalidOperationException>(res);
}

[Test]
public void GetEnumerator_ListWith3Elements_AllAreCorrect()
{
    CustomLinkedList<string>? list = new CustomLinkedList<string>();

    Node<string> node1 = new("node1");
    Node<string> node2 = new("node2");
    Node<string> node3 = new("node3");

    _list.AddLast(node1);
    _list.AddLast(node2);
    _list.AddLast(node3);

    foreach(var item in _list)
```

```
        {
            list.Add(item);
        }

        Assert.That(list, Is.EqualTo(_list));
    }
}
```

Результати виконання тестів:



Покриття тестами коду:



Covered	Uncovered	Coverable	Total	Line coverage	Covered	Total	Branch coverage
322	0	322	505	100%	4	4	100%
208	23	231	363	90%	80	92	86.9%
195	3	198	296	98.4%	78	90	86.6%
0	20	20	37	0%	0	0	
13	0	13	30	100%	2	2	100%

Висновок: у ході лабораторної роботи як познайомився з програмними компонентами та засобами тестування програмного коду, навчився писати юніт-тести та ознайомився з поняттям покриття коду.