

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

Лабораторна робота №2  
з дисципліни «Сучасні технології розробки WEB-застосунків на  
платформі Microsoft.NET»  
на тему: «Модульне тестування. Ознайомлення з засобами та  
практиками модульного тестування»

Виконав:  
студент гр. ІС-11  
Іваніцький М. К.  
Викладач:  
Бардін В.

**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.

**Лістинг програмного коду:**

**MyLinkedListTests.cs**

```
namespace MyCollectionTesting
{
    public abstract class Tests
    {
        public MyLinkedList<int> _list;
        [SetUp]
        public void Setup()
        {
            _list = new MyLinkedList<int>();
        }
    }

    [TestFixture]
    public class ConstructorTests
    {
        [Test]
        public void
        ConstructorWithCollection_NormalCollection_ResultListIsEqualToInitial()
        {
            List<int> oldList = new List<int>() { 2, 3, 4 };
            var newList = new MyLinkedList<int>(oldList);
        }
    }
}
```

```

        Assert.Multiple(() =>
        {
            Assert.That(newList.Count, Is.EqualTo(3));
            Assert.That(newList.First!.item, Is.EqualTo(2));
            Assert.That(newList.Last!.item, Is.EqualTo(4));
            Assert.That(newList.Last.prev!.item, Is.EqualTo(3));
        });
    }
}

[TestFixture]
public class AddLastTests : Tests
{
    [Test]
    public void Add_NewValue_EmptyList_ResultListWithOneNode()
    {
        var value = 5;
        _list.Add(value);
        Assert.Multiple(() =>
        {
            Assert.That(_list.Last!.item, Is.EqualTo(value));
            Assert.That(_list.Count, Is.EqualTo(1));
        }
        );
    }

    [Test]
    public void Add_NewValue_ListWithOneNode_ResultLastValueIsNewValue()
    {
        var value = 5;
        _list.Add(value);
        var value1 = 7;
        _list.Add(value1);
        Assert.Multiple(() =>
        {
            Assert.That(_list.Last!.item, Is.EqualTo(value1));
            Assert.That(_list.Count, Is.EqualTo(2));
        }
        );
    }
}

```

```

    }
}

[TestFixture]
public class AddFirstTests : Tests
{
    [Test]
    public void AddFirst_NewValue_EmptyList_ResultListWithOneNode()
    {
        var value = 5;
        _list.AddFirst(value);
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.item, Is.EqualTo(value));
            Assert.That(_list.Count, Is.EqualTo(1));
        });
    }

    [Test]
    public void AddFirst_NewValue_ListWithOneNode_ResultFirstIsNewValue()
    {
        var value = 5;
        _list.AddFirst(value);
        var value1 = 7;
        _list.AddFirst(value1);
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.item, Is.EqualTo(value1));
            Assert.That(_list.Count, Is.EqualTo(2));
        });
    }
}

[TestFixture]
public class AddAfterTests : Tests
{
    [Test]

```

```

        public void
AddAfter_NewValue_BetweenTwoValues_ResultNewValueInItsPlace()
    {
        var value = 5;
        _list.Add(value);
        var value1 = 7;
        _list.Add(value1);
        var value2 = 6;
        _list.AddAfter(_list.First!, value2);
        var node = _list.Find(value2);
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.next, Is.EqualTo(node));
            Assert.That(node!.prev, Is.EqualTo(_list.First));
            Assert.That(_list.Last!.prev, Is.EqualTo(node));
            Assert.That(node!.next, Is.EqualTo(_list.Last));
            Assert.That(_list.Count, Is.EqualTo(3));
        });
    }

[Test]
public void AddAfter_NewValue_AfterLast_ResultTheLastIsNewValue()
{
    var value = 5;
    _list.Add(value);
    var value1 = 7;
    _list.AddAfter(_list.Last!, value1);
    var node = _list.Find(7);
    Assert.Multiple(() =>
    {
        Assert.That(_list.First!.next, Is.EqualTo(node));
        Assert.That(node!.prev, Is.EqualTo(_list.First));
        Assert.That(node, Is.EqualTo(_list.Last));
        Assert.That(node!.next, Is.EqualTo(null));
        Assert.That(_list.Count, Is.EqualTo(2));
    });
}

```

```

[Test]
public void AddAfter_NewValue_AfterNull_ResultException()
{
    var node = _list.Last;
    var value = 5;
    Assert.Throws<ArgumentNullException>(() => _list.AddAfter(node,
value));
}

[Test]
public void AddAfter_NewValue_FakeNode_ResultException()
{
    var node = new Node<int>(5);
    var value = 7;
    Assert.Throws<InvalidOperationException>(() =>
_list.AddAfter(node, value));
}

[TestFixture]
public class AddBeforeTests : Tests
{
    [Test]
    public void
AddBefore_NewValue_BetweenTwoValues_ResultNewValueInItsPlace()
    {
        var value = 5;
        _list.Add(value);
        var value1 = 7;
        _list.Add(value1);
        var value2 = 6;
        _list.AddBefore(_list.Last, value2);
        var node = _list.Find(value2);
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.next, Is.EqualTo(node));
            Assert.That(node!.prev, Is.EqualTo(_list.First));
            Assert.That(_list.Last!.prev, Is.EqualTo(node));
            Assert.That(node!.next, Is.EqualTo(_list.Last));
        }
    }
}

```

```

        Assert.That(_list.Count, Is.EqualTo(3));
    });
}

[Test]
public void AddBefore_NewValue_BeforeFirst_ResultTheFirstIsNewValue()
{
    var value = 5;
    _list.Add(value);
    var value1 = 7;
    _list.AddBefore(_list.First, value1);
    var node = _list.Find(7);
    Assert.Multiple(() =>
    {
        Assert.That(_list.Last!.prev, Is.EqualTo(node));
        Assert.That(node!.next, Is.EqualTo(_list.Last));
        Assert.That(node, Is.EqualTo(_list.First));
        Assert.That(node!.prev, Is.EqualTo(null));
        Assert.That(_list.Count, Is.EqualTo(2));
    });
}

[Test]
public void AddBefore_NewValue_AfterNull_ResultException()
{
    var node = _list.Last;
    var value = 5;
    Assert.Throws<ArgumentNullException>(() => _list.AddBefore(node,
value));
}

[Test]
public void AddBefore_NewValue_FakeNode_ResultException()
{
    var node = new Node<int>(5);
    var value = 7;
    Assert.Throws<InvalidOperationException>(() =>
_list.AddBefore(node, value));
}

```

```

    }
}

[TestFixture]
public class RemoveTests : Tests
{
    [Test]
    public void Remove_EmptyList_ResultException()
    {
        Assert.That(_list.Remove(0), Is.EqualTo(false));
    }

    [Test]
    public void Remove_BetweenTwoNodes_ResultValueRemoved()
    {
        var value = 5;
        var value1 = 6;
        var value2 = 7;
        _list.Add(value);
        _list.Add(value1);
        _list.Add(value2);
        _list.Remove(6);
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.next, Is.EqualTo(_list.Last));
            Assert.That(_list.Last!.prev, Is.EqualTo(_list.First));
            Assert.That(_list.Count, Is.EqualTo(2));
        });
    }

    [Test]
    public void Remove_FakeNode_ResultException()
    {
        var node = new Node<int>(5);
        Assert.Throws<InvalidOperationException>(() =>
            _list.Remove(node));
    }
}

```



```

[Test]
public void Remove_OnlyNode_ResultEmptyList()
{
    var node = new Node<int>(5);
    _list.AddLast(node);
    _list.Remove(node);
    Assert.That(_list.Count, Is.EqualTo(0));
}

[TestFixture]
public class RemoveFirstTests : Tests
{
    [Test]
    public void RemoveFirst_EmptyList_ResultException()
    {
        Assert.Throws<InvalidOperationException>(() =>
            _list.RemoveFirst());
    }

    [Test]
    public void RemoveFirst_TwoElements_ResultSecondBecomeFirst()
    {
        var value = 5;
        var value1 = 7;
        _list.AddLast(value);
        _list.AddLast(value1);
        _list.RemoveFirst();
        Assert.Multiple(() =>
        {
            Assert.That(_list.First!.item, Is.EqualTo(value1));
            Assert.That(_list.Count, Is.EqualTo(1));
        });
    }
}

[TestFixture]
public class RemoveLastTests : Tests

```

```

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

    [Test]
    public void RemoveLast_TwoElements_ResultFirstBecomeLast()
    {
        var value = 5;
        var value1 = 7;
        _list.AddLast(value);
        _list.AddLast(value1);
        _list.RemoveLast();
        Assert.Multiple(() =>
        {
            Assert.That(_list.Last!.item, Is.EqualTo(value));
            Assert.That(_list.Count, Is.EqualTo(1));
        });
    }
}

[TestFixture]
public class ClearTests : Tests
{
    [Test]
    public void Clear_ListWith4Elements_ResultListIsEmpty()
    {
        for(int i = 0; i < 4; i++)
        {
            _list.AddLast(i);
        }
        _list.Clear();
        Assert.Multiple(() =>
        {
            Assert.That(_list.Count, Is.EqualTo(0));
        });
    }
}

```

```

        Assert.That(_list.Last, Is.Null);
        Assert.That(_list.First, Is.Null);
    });
}

[TestFixture]
public class FindTests : Tests
{
    [Test]
    public void Find_EmptyList_ResultIsNull()
    {
        var node = _list.Find(2);
        Assert.That(node, Is.Null);
    }
}

[TestFixture]
public class ContainsTests : Tests
{
    [Test]
    public void Contains_ResultIsTrue()
    {
        var value = 5;
        _list.AddLast(value);
        Assert.IsTrue(_list.Contains(value));
    }
}

[TestFixture]
public class CopyToTests : Tests
{
    [Test]
    public void
CopyTo_ListWith3ElementsAndValidIndexParameter_ResultArrayWith3Elements()
    {
        int[] arr = new int[3];

```

```

        Node<int> node1 = new(2);
        Node<int> node2 = new(3);
        Node<int> node3 = new(4);

        _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
CopyTo_ListWith3ElementsAndWrongIndexInParams_ThrowArgumentOutOfRangeException()
    {
        int[] arr = new int[3];

        Node<int> node1 = new(2);
        Node<int> node2 = new(3);
        Node<int> node3 = new(4);

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

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

        Assert.Throws<ArgumentOutOfRangeException>(res);
    }
}

[TestFixture]

```

```

public class GetEnumeratorTests : Tests
{
    [Test]
    public void GetEnumerator_ListWith3Elements_AllAreCorrect()
    {
        MyLinkedList<int>? list = new MyLinkedList<int>();

        Node<int> node1 = new(2);
        Node<int> node2 = new(3);
        Node<int> node3 = new(4);

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

        foreach (var item in _list)
        {
            list.Add(item);
        }

        Assert.That(list.First!.item, Is.EqualTo(_list.First!.item));
        Assert.That(list.Last!.item, Is.EqualTo(_list.Last!.item));
    }
}

```

## EventListenerTests.cs

```

[TestFixture]
public class EventListenerTests
{
    private EventListener<int> _eventListener;
    private MyLinkedList<int> _list;
    [SetUp]
    public void Setup()
    {
        _eventListener = new EventListener<int>();
        _list = new MyLinkedList<int>();
    }

    [Test]
    public void OnAddedHandler_AddHandler_TureReturn()
    {
        var wasCalled = false;

        _list.ItemAdded += _eventListener.OnAdded!;
        _list.ItemRemoved += _eventListener.OnRemoved!;
    }
}

```

```

        _list.ItemAdded += (o, e) => wasCalled = true;

        Node<int> node = new(2);
        Node<int> node1 = new(3);
        _list.AddLast(node);
        _list.AddLast(node1);

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

    [Test]
    public void OnRemovedHandler_AddHandler_TureReturn()
    {
        var wasCalled = false;

        _list.ItemAdded += _eventListener.OnAdded!;
        _list.ItemRemoved += _eventListener.OnRemoved!;

        _list.ItemAdded += (o, e) => wasCalled = true;

        Node<int> node = new(2);
        Node<int> node1 = new(3);
        _list.AddLast(node);
        _list.AddLast(node1);
        _list.RemoveLast();

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

```

### Результат виконання тестів:

Поиск (Ctrl+I)		
Запуск тестов завершён ⚠ Предупреждение ❌ Ошибка		
Тестирование	Длительность...	Признак
▲ ✓ MyCollectionT...	130 мс	
▲ ✓ MyCollection...	130 мс	
▸ ✓ AddAfterTe...	46 мс	
▸ ✓ AddBefore...	11 мс	
▸ ✓ AddFirstTe...	< 1 мс	
▸ ✓ AddLastTests	< 1 мс	
▸ ✓ ClearTests (...)	< 1 мс	
▸ ✓ Constructo...	< 1 мс	
▸ ✓ ContainsTe...	< 1 мс	
▸ ✓ CopyToTests	3 мс	
▸ ✓ EventListen...	1 мс	
▸ ✓ FindTests (1)	< 1 мс	
▸ ✓ GetEnumerator...	< 1 мс	
▸ ✓ RemoveFir...	< 1 мс	
▸ ✓ RemoveLas...	< 1 мс	
▸ ✓ RemoveTests	2 мс	
▸ ✓ ShowTests (..	67 мс	

## Покриття тестами коду:

▼ Name
– MyCollection
MyCollection.MyLinkedList`1
MyCollection.Node`1
MyCollection.ItemEventArgs`1
lab1.EventListener`1

▼ Covered	▲ Uncovered	▼ Coverable	▼ Total	▼ Line coverage		▼ Covered	▼ Total	▼ Branch coverage	
249	6	255	384	97.6%	<div><div></div></div>	64	70	91.4%	<div><div></div></div>
227	6	233	319	97.4%	<div><div></div></div>	58	62	93.5%	<div><div></div></div>
7	0	7	23	100%	<div><div></div></div>	0	0		<div><div></div></div>
7	0	7	19	100%	<div><div></div></div>	0	0		<div><div></div></div>
8	0	8	23	100%	<div><div></div></div>	6	8	75%	<div><div></div></div>

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