

Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №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 групи ІС-11

Ходос Михайло

Київ – 2023

Зміст

1.	Постановка задачі.....	3
2.	Лістинг коду	5
3.	Результати виконання	<u>6</u>
4.	Висновок	<u>6</u>

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 коду

Посилання на GitHub репозиторій:

<https://github.com/Mechano1d/.Net/tree/Lab2>

Клас Element Tests

```
using MyCollection;

namespace Lab2
{
    [TestFixture]
    public class ElementTests
    {
        private Fixture _fixture;
        private _SortedList<int, int> MySortedList;
        private SortedList<int, int> TestList;

        [SetUp]
        public void Setup()
        {
            _fixture = new Fixture();
            MySortedList = new _SortedList<int, int>(4);
        }

        [Test]
        public void ElementTest_Add_Empty()
        {
            TestList = new SortedList<int, int>();
            var a = _fixture.Create<KeyValuePair<int, int>>();
            MySortedList.Add(a);
            TestList.Add(a.Key, a.Value);
            for (int i = 0; i < MySortedList.Count(); i++)
            {
                Assert.That(MySortedList[i].Key,
                    Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
                Assert.That(MySortedList[i].Value,
                    Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
            }
        }

        [Test]
        public void ElementTest_Add_Duplicate_KeyValuePair()
        {
            TestList = _fixture.Create<SortedList<int, int>>();
            foreach (var n in TestList)
            {
                MySortedList.Add(n.Key, n.Value);
            }
            var a = MySortedList[0];

            Assert.Throws<ArgumentException>(() => MySortedList.Add(a));
            Assert.Throws<ArgumentException>(() => TestList.Add(a.Key, a.Value));
            for (int i = 0; i < MySortedList.Count(); i++)
            {
                Assert.That(MySortedList[i].Key,
                    Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
                Assert.That(MySortedList[i].Value,
                    Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
            }
        }
    }
}
```

```

    }
    [Test]
    public void ElementTest_Add_Duplicate_KeyAndValue()
    {
        TestList = _fixture.Create<SortedList<int, int>>();
        foreach (var n in TestList)
        {
            MySortedList.Add(n.Key, n.Value);
        }

        var a = MySortedList[0];

        Assert.Throws<ArgumentException>(() => MySortedList.Add(a.Key,
a.Value));
        Assert.Throws<ArgumentException>(() => TestList.Add(a.Key, a.Value));
        for (int i = 0; i < MySortedList.Count(); i++)
        {
            Assert.That(MySortedList[i].Key,
Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
            Assert.That(MySortedList[i].Value,
Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
        }
    }
    [Test]
    public void ElementTest_Add_OverCapacity_KeyValuePair()
    {
        TestList = new SortedList<int, int>(4);
        MySortedList = new _SortedList<int, int>(4);

        for (int i = 0; i < 6; i++)
        {
            var a = _fixture.Create<KeyValuePair<int, int>>();
            TestList.Add(a.Key, a.Value);
            MySortedList.Add(a);
        }

        for (int i = 0; i < MySortedList.Count(); i++)
        {
            Assert.That(MySortedList[i].Key,
Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
            Assert.That(MySortedList[i].Value,
Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
        }
    }
    [Test]
    public void ElementTest_Add_OverCapacity_KeyAndValue()
    {
        TestList = new SortedList<int, int>(4);
        MySortedList = new _SortedList<int, int>(4);

        for (int i = 0; i < 6; i++)
        {
            var a = _fixture.Create<KeyValuePair<int, int>>();
            TestList.Add(a.Key, a.Value);
            MySortedList.Add(a.Key, a.Value);
        }

        for (int i = 0; i < MySortedList.Count(); i++)
        {
            Assert.That(MySortedList[i].Key,
Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
            Assert.That(MySortedList[i].Value,
Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
        }
    }

```

```

[Test]
public void ElementTest_Add_KeyValuePair()
{
    TestList = _fixture.Create<SortedList<int, int>>();
    foreach (var n in TestList)
    {
        MySortedList.Add(n.Key, n.Value);
    }
    var a = _fixture.Create<KeyValuePair<int, int>>();
    MySortedList.Add(a);
    TestList.Add(a.Key, a.Value);
    for (int i = 0; i < MySortedList.Count(); i++)
    {
        Assert.That(MySortedList[i].Key,
            Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
        Assert.That(MySortedList[i].Value,
            Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
    }
}

[Test]
public void ElementTest_Add_KeyAndValue()
{
    TestList = _fixture.Create<SortedList<int, int>>();
    foreach (var n in TestList)
    {
        MySortedList.Add(n.Key, n.Value);
    }
    var key = _fixture.Create<int>();
    var value = _fixture.Create<int>();

    MySortedList.Add(key, value);
    TestList.Add(key, value);
    for (int i = 0; i < MySortedList.Count(); i++)
    {
        Assert.That(MySortedList[i].Key,
            Is.EqualTo(TestList.Keys[MySortedList.Count() - i - 1]));
        Assert.That(MySortedList[i].Value,
            Is.EqualTo(TestList.Values[MySortedList.Count() - i - 1]));
    }
}

[Test]
public void ElementTest_Contains_KeyValuePair()
{
    TestList = _fixture.Create<SortedList<int, int>>();
    foreach (var n in TestList)
    {
        MySortedList.Add(n.Key, n.Value);
    }
    var InvalidElement = new KeyValuePair<int, int>(-1, -1);

    bool ContainsTrue = MySortedList.Contains(MySortedList[0]);
    bool ContainsFalse = MySortedList.Contains(InvalidElement);

    Assert.That(ContainsTrue, Is.True);
    Assert.That(ContainsFalse, Is.False);
}

[Test]
public void ElementTest_Contains_Key()
{
    TestList = _fixture.Create<SortedList<int, int>>();
    foreach (var n in TestList)
    {
        MySortedList.Add(n.Key, n.Value);
    }
}

```

```

        bool ContainsTrue = MySortedList.Contains(MySortedList[0].Key);
        bool ContainsFalse = MySortedList.Contains(-1);

        Assert.That(ContainsTrue, Is.True);
        Assert.That(ContainsFalse, Is.False);
    }
    [Test]
    public void ElementTest_Remove_KeyValuePair_Head()
    {
        TestList = _fixture.Create<SortedList<int, int>>();
        foreach (var n in TestList)
        {
            MySortedList.Add(n.Key, n.Value);
        }
        var InvalidElement = new KeyValuePair<int, int>(-1, -1);

        bool ContainsTrue = MySortedList.Remove(MySortedList[0]);
        bool ContainsFalse = MySortedList.Remove(InvalidElement);

        Assert.That(ContainsTrue, Is.True);
        Assert.That(ContainsFalse, Is.False);
    }
    [Test]
    public void ElementTest_Remove_Key_Head()
    {
        TestList = _fixture.Create<SortedList<int, int>>();
        foreach (var n in TestList)
        {
            MySortedList.Add(n.Key, n.Value);
        }

        bool ContainsTrue = MySortedList.Remove(MySortedList[0].Key);
        bool ContainsFalse = MySortedList.Remove(-1);

        Assert.That(ContainsTrue, Is.True);
        Assert.That(ContainsFalse, Is.False);
    }
    [Test]
    public void ElementTest_Remove_KeyValuePair_Middle()
    {
        TestList = _fixture.Create<SortedList<int, int>>();
        foreach (var n in TestList)
        {
            MySortedList.Add(n.Key, n.Value);
        }
        var InvalidElement = new KeyValuePair<int, int>(-1, -1);

        bool ContainsTrue = MySortedList.Remove(MySortedList[MySortedList.size /
2]);

        Assert.That(ContainsTrue, Is.True);
    }
    [Test]
    public void ElementTest_Remove_Key_Middle()
    {
        TestList = _fixture.Create<SortedList<int, int>>();
        foreach (var n in TestList)
        {
            MySortedList.Add(n.Key, n.Value);
        }

        bool ContainsTrue = MySortedList.Remove(MySortedList[MySortedList.size /
2].Key);

```

```

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

```

Клас ArrayTests

```
using MyCollection;
```

```
namespace Lab2
```

```

{
    [TestFixture]
    public class ArrayTests
    {
        private Fixture _fixture;
        private _SortedList<int, int> MySortedList;
        private SortedList<int, int> TestList;

        [SetUp]
        public void Setup()
        {
            _fixture = new Fixture();
            MySortedList = new _SortedList<int, int>(4);
            TestList = _fixture.Create<SortedList<int, int>>();
            foreach (var n in TestList)
            {
                MySortedList.Add(n);
            }
        }

        [Test]
        public void ArrayTest_SameSizeArray()
        {
            var SameSizeArray = new int[MySortedList.size];

            MySortedList.CopyTo(SameSizeArray, 0);
            int l = 0;

            for (int i = 0; i < MySortedList.Count; i++)
            {
                Assert.That(MySortedList[i].Value, Is.EqualTo(SameSizeArray[l]));
                l++;
            }
        }

        [Test]
        public void ArrayTest_DifferentSizeArray_NoOffset()
        {
            var IndexOffset = _fixture.Create<int>();
            IndexOffset %= MySortedList.Count;
            var SameSizeArray = new int[MySortedList.Count + IndexOffset];

            MySortedList.CopyTo(SameSizeArray, 0);
            int l = 0;

            for (int i = 0; i < MySortedList.Count; i++)
            {
                Assert.That(MySortedList[i].Value, Is.EqualTo(SameSizeArray[l]));
                l++;
            }
        }

        [Test]
        public void ArrayTest_DifferentSizeArray_Offset()

```



```

    {
        var IndexOffset = _fixture.Create<int>();
        IndexOffset %= MySortedList.Count;
        var SameSizeArray = new int[MySortedList.Count + IndexOffset];

        MySortedList.CopyTo(SameSizeArray, IndexOffset);
        int l = IndexOffset;

        for (int i = 0; i < MySortedList.Count; i++)
        {
            Assert.That(MySortedList[i].Value, Is.EqualTo(SameSizeArray[l]));
            l++;
        }
    }
    [Test]
    public void ArrayTest_ArrayTooSmall()
    {
        var SmallArray = new int[MySortedList.Count - 1];

        Assert.Throws<ArgumentException>(() => MySortedList.CopyTo(SmallArray,
0));
    }
    [Test]
    public void ArrayTest_InvalidIndex()
    {
        var Array = new int[MySortedList.Count];

        Assert.Throws<ArgumentOutOfRangeException>(() =>
MySortedList.CopyTo(Array, -1));
    }
    [Test]
    public void ArrayTest_NullArray()
    {
        int[] NullArray = null;

        Assert.Throws<ArgumentNullException>(() =>
MySortedList.CopyTo(NullArray, 0));
    }
}
}

```

Клас ConstructorTests

```

using MyCollection;

namespace Lab2
{
    [TestFixture]
    public class ConstructorTests
    {
        [Test]
        public void ConstructorTest_ArgumentNull()
        {
            Assert.Throws<ArgumentNullException>(() => new _SortedList<object,
object>(null));
        }
        [Test]
        public void ConstructorTest_ArgumentNegative()
        {
            Assert.Throws<ArgumentOutOfRangeException>(() => new _SortedList<object,
object>(-1));
        }
    }
}

```

```

        public void ConstructorTest_ArgumentZero()
        {
            var MyList = new _SortedList<object, object>(0);
            Assert.That(MyList._items.Length, Is.EqualTo(0));
        }
        [Test]
        public void ConstructorTest_ArgumentValid()
        {
            var MyList = new _SortedList<object, object>(10);
            Assert.That(MyList._items.Length, Is.EqualTo(10));
        }
    }
}

```

Клас EventTests

```

using MyCollection;

namespace Lab2
{
    [TestFixture]
    public class EventTests
    {
        private Fixture _fixture;
        private _SortedList<int, int> MySortedList;
        private SortedList<int, int> TestList;

        [SetUp]
        public void Setup()
        {
            _fixture = new Fixture();
            MySortedList = new _SortedList<int, int>(4);
        }

        [Test]
        public void EventTest_ElementAdded()
        {
            bool EventTriggered = false;
            var element = _fixture.Create<KeyValuePair<int, int>>();

            MySortedList.AddElement += (sender, e) => EventTriggered = true;
            MySortedList.Add(element);

            Assert.IsTrue(EventTriggered);
        }

        [Test]
        public void EventTest_ElementRemoved()
        {
            bool EventTriggered = false;
            var element = _fixture.Create<KeyValuePair<int, int>>();

            MySortedList.RemoveElement += (sender, e) => EventTriggered = true;
            MySortedList.Add(element);
            MySortedList.Remove(element);

            Assert.IsTrue(EventTriggered);
        }
    }
}

```

```

[Test]
public void EventTest_ClearArray()
{
    bool EventTriggered = false;
    TestList = _fixture.Create<SortedList<int, int>>();
    foreach (var n in TestList)
    {
        MySortedList.Add(n.Key, n.Value);
    }

    MySortedList.ClearArray += (sender, e) => EventTriggered = true;
    MySortedList.Clear();

    Assert.IsTrue(EventTriggered);
}
}
}

```

Клас EnumeratorTests

```

using MyCollection;

namespace Lab2
{
    [TestFixture]
    public class EnumeratorTests
    {
        private Fixture _fixture;
        private _SortedList<int, int> MySortedList;
        private SortedList<int, int> TestList;

        [SetUp]
        public void Setup()
        {
            _fixture = new Fixture();
            MySortedList = new _SortedList<int, int>(4);
        }

        [Test]
        public void EventTest_ElementAdded()
        {
            TestList = _fixture.Create<SortedList<int, int>>();
            foreach (var n in TestList)
            {
                MySortedList.Add(n.Key, n.Value);
            }

            int l = 0;
            var MyEnum = MySortedList.GetEnumerator();

            while(MyEnum.MoveNext())
            {
                Assert.That(MyEnum.Current.Key,
                    Is.EqualTo(TestList.Keys[TestList.Count - l - 1]));
                Assert.That(MyEnum.Current.Value,
                    Is.EqualTo(TestList.Values[TestList.Count - l - 1]));
                l++;
            }
        }
    }
}

```

```

[Test]
public void EventTest_Reset()
{
    MySortedList = _fixture.Create<_SortedList<int, int>>();

    var MyEnum = MySortedList.GetEnumerator();
    bool NonZeroCheck;

    while (MyEnum.MoveNext())
    {}
    NonZeroCheck = (MyEnum.Current.Value != MySortedList[0].Value &&
MyEnum.Current.Key != MySortedList[0].Key);
    MyEnum.Reset();

    Assert.That(NonZeroCheck, Is.EqualTo(true));
    Assert.That(MyEnum.Current, Is.EqualTo(MySortedList[0]));
}
}
}

```

Клас ListWideTests

```

using MyCollection;

namespace Lab2
{
    [TestFixture]
    public class ListWideTests
    {
        private Fixture _fixture;
        private _SortedList<int, int> MySortedList;
        private _SortedList<int, int> MySortedList_Initial;
        private _SortedList<int, int> MySortedList_Final;
        private SortedList<int, int> TestList;

        [SetUp]
        public void Setup()
        {
            _fixture = new Fixture();
        }

        [Test]
        public void ListWideTest_Clear()
        {
            MySortedList = _fixture.Create<_SortedList<int, int>>();

            Assert.That(MySortedList.Count, Is.GreaterThan(0));
            MySortedList.Clear();
            Assert.That(MySortedList.Count, Is.EqualTo(0));
        }

        [Test]
        public void ListWideTest_ExpandCapacity()
        {
            var InitialCapacity = _fixture.Create<int>();
            MySortedList_Initial = new _SortedList<int, int>(InitialCapacity);
            MySortedList_Final = MySortedList_Initial;

            bool InitialCapacityCheck = (InitialCapacity ==
MySortedList_Initial._items.Length);
            MySortedList_Final.ExpandCapacity();
        }
    }
}

```

```

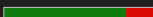
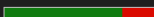


        bool FinalCapacityCheck = (InitialCapacity * 2 ==
MySortedList_Final._items.Length);
        int l = 0;

        Assert.That(InitialCapacityCheck, Is.True);
        Assert.That(FinalCapacityCheck, Is.True);

        foreach(var n in MySortedList_Initial)
        {
            Assert.That(n, Is.EqualTo(MySortedList_Final[l]));
            l++;
        }
    }
}
}

```

Результати виконання

	▼ Covered	▼ Uncovered	▼ Coverable	▼ Total	▼ Line coverage	▼ Covered	▼ Total	▼ Branch coverage
– MyCollection	303	20	323	430	93.8% 	94	104	90.3% 
MyCollection_SortedList'2	303	20	323	430	93.8% 	94	104	90.3% 

Висновок

На даній лабораторній роботі я використав інструменти фреймворку NUnit для модульного тестування попередньо створеної колекції. Покриття тестами складає близько 93%.