МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТУКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Лабораторна робота №2

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

на тему: «Модульне тестування. Ознайомлення з засобами та практиками модульного тестування»

Викладач:

Бардін В.

Виконав студент:

Лазюта Олексій

групи ІП-15 ФІОТ

Київ-2023

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.

Хід роботи:

1. Код програми

Файл EnumeratorTest.cs:

```
using System.Collections;

namespace MyLibrary;

public class CustomQueue <T>: IEnumerable<T> where T: IComparable
{
    private int _size;
    private Node<T>? _headNode;
    private Node<T>? _tailNode;

    public event EventHandler? OnEnqueued;
    public event EventHandler? OnDequeued;
    public event EventHandler? OnCleared;
    public event EventHandler? OnCopiedTo;
    public event EventHandler? OnReversed;

    public int Count => _size;

    public CustomQueue()
    {
        _headNode = null;
        _tailNode = null;
        _size= 0;
    }

    public CustomQueue(IEnumerable<T> collection)
    {
        _headNode = null;
        _tailNode = null;
        _size = 0;
        foreach (var item in collection)
        {
            Enqueue(item);
        }
    }

    IEnumerator<T> IEnumerable<T>.GetEnumerator()
    {
        Node<T>? currentNode = _headNode;
        while (currentNode != null)
        {
            yield return currentNode.Value;
            currentNode = currentNode.Next;
        }
    }
```

```
    public IEnumerator GetEnumerator()
    {
        return new MyEnumerator(_headNode);
    }

    public void CopyTo(T[]? array, int index)
    {
        Node<T>? activeNode = _headNode;
        if (array == null)
            throw new ArgumentNullException("Value can not be null");
        else if (index < 0 || index > array.Length - 1)
            throw new ArgumentOutOfRangeException("Index can not be out of
range");
        else if (array.Length - index < _size)
            throw new ArgumentException("Number of the elements in CustomQueue
is greater than the Array can contain");

        while (activeNode != null)
        {
            array[index] = activeNode.Value;
            activeNode = activeNode.Next;
            index++;
        }
        OnCopiedTo?.Invoke(this, EventArgs.Empty);
    }

    public void Enqueue(T item)
    {
        Node<T>? node = new Node<T>(item);
        if (_headNode == null)
        {
            _headNode = node;
            _tailNode = node;
        }

        else
        {
            _tailNode.Next = new Node<T>(item);
            _tailNode = _tailNode.Next;
        }

        _size++;
        OnEnqueued?.Invoke(this, new EventArgs());
    }

    public T Dequeue()
    {
        if (_headNode == null)
            throw new InvalidOperationException("CustomQueue is empty");

        T headNodeValue = _headNode.Value;
        _headNode = _headNode.Next;

        _size--;
        OnDequeued?.Invoke(this, EventArgs.Empty);
        return headNodeValue;

    }

    public void Clear()
    {
        _headNode = null;
        _tailNode = null;
        OnCleared?.Invoke(this, EventArgs.Empty);
    }
```

```
    public bool Contains(T item)
    {
        Node<T>? activeNode = _headNode;
        while (activeNode != null)
        {
            if (activeNode.Value.Equals(item))
                return true;
            activeNode = activeNode.Next;
        }

        return false;
    }

    public T Peek()
    {
        if (_size == 0)
            throw new InvalidOperationException("CustomQueue is empty");

        return _headNode.Value;
    }

    public T[] ToArray()
    {
        Node<T>? activeNode = _headNode;
        T[] array = new T[_size];
        for (int i = 0; i < _size; i++)
        {
            array[i] = activeNode.Value;
            activeNode = activeNode.Next;
        }

        return array;
    }

    public CustomQueue<T> Reverse()
    {
        CustomQueue<T> reversedQueue;
        IEnumerable<T> tempArray;
        tempArray = this.ToArray().Reverse();
        reversedQueue = new CustomQueue<T>(tempArray);
        OnReversed?.Invoke(this, EventArgs.Empty);
        return reversedQueue;
    }

    private class MyEnumerator: IEnumerator<T>
    {
        private Node<T>? _activeNode;
        private Node<T>? _headNode;
        private bool _beganEnumerating = false;

        public T Current => _activeNode.Value;

        object IEnumerator.Current => Current;

        public MyEnumerator(Node<T>? headNode)
        {
            _headNode = headNode;
            _activeNode = _headNode;
        }

        public bool MoveNext()
        {
            if (_headNode == null)
                return false;
```

```
            else if(!_beganEnumerating)
            {
                _beganEnumerating = true;
                _activeNode = _headNode;
                return true;
            }

            bool hasNext = _activeNode.Next != null;
            if (!hasNext)
            {
                return false;
            }

            _activeNode = _activeNode.Next;
            return hasNext;
        }

        public void Reset()
        {
            _activeNode = _headNode;
        }

        public void Dispose()
        {

        }
    }
}
```

Файл EventsTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class EventsTest
{
    [Fact]
    public void Enqueue_OnCall_Invoke()
    {
        CustomQueue<int> cq = new CustomQueue<int>();
        bool wasRaised = false;
        int element = 1;
        object senderClass = null;

        cq.OnEnqueued += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };
        cq.Enqueue(element);

        Assert.True(wasRaised);
        Assert.IsType<CustomQueue<int>>(senderClass);
    }

    [Fact]
    public void Dequeue_OnCall_Invoke()
    {
        CustomQueue<int> cq = new CustomQueue<int>();
        bool wasRaised = false;
        int element = 1;
        object senderClass = null;

        cq.Enqueue(element);
        cq.OnDequeued += (sender, args) =>
```

```
        {
            wasRaised = true;
            senderClass = sender;
        };
        cq.Dequeue();

        Assert.True(wasRaised);
        Assert.IsType<CustomQueue<int>>(senderClass);
    }

    [Fact]
    public void Clear_OnCall_Invoke()
    {
        CustomQueue<int> cq = new CustomQueue<int>(new int[] {2, 41, 1234, -1})
;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnCleared += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };
        cq.Clear();

        Assert.True(wasRaised);
        Assert.IsType<CustomQueue<int>>(senderClass);
    }

    [Fact]
    public void Reverse_OnCall_Invoke()
    {
        CustomQueue<int> cq = new CustomQueue<int>(new int[] {2, 41, 1234, -1})
;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnReversed += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };
        cq.Reverse();

        Assert.True(wasRaised);
        Assert.IsType<CustomQueue<int>>(senderClass);
    }

    [Theory]
    [MemberData(nameof(SuccessfulCopyToEventTestData))]
    public void CopyTo_OnCall_Invoke(CustomQueue<int> startingCollection, int[]
array, int index)
    {
        CustomQueue<int> cq = startingCollection ;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnCopiedTo += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };
        cq.CopyTo(array, index);

        Assert.True(wasRaised);
        Assert.IsType<CustomQueue<int>>(senderClass);
```

```
    }

    [Theory]
    [MemberData(nameof(OutOfRangeCopyToEventTestData))]
    public void CopyTo_OutOfRangeException_DontInvoke(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection ;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnCopiedTo += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };

        Assert.Throws<ArgumentOutOfRangeException>(()=>cq.CopyTo(array, index));
        Assert.False(wasRaised);
    }

    [Theory]
    [MemberData(nameof(NullExceptionCopyToEventTestData))]
    public void CopyTo_NullExceptionException_DontInvoke(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection ;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnCopiedTo += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };

        Assert.Throws<ArgumentNullException>(()=>cq.CopyTo(array, index));
        Assert.False(wasRaised);
    }

    [Theory]
    [MemberData(nameof(ArgumentExceptionCopyToEventTestData))]
    public void CopyTo_ArgumentException_DontInvoke(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection ;
        bool wasRaised = false;
        object senderClass = null;

        cq.OnCopiedTo += (sender, args) =>
        {
            wasRaised = true;
            senderClass = sender;
        };

        Assert.Throws<ArgumentException>(()=>cq.CopyTo(array, index));
        Assert.False(wasRaised);
    }
    public static IEnumerable<object[]> SuccessfulCopyToEventTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] { 2, 4, 5, 5, 6},
            2
        };
        yield return new object[] {
```

```
            new CustomQueue<int>(new int[] { 0, 0, 0, 1}),
            new int[] { 24, 451, -5, 85, 6},
            0
        };
    }

    public static IEnumerable<object[]> OutOfRangeCopyToEventTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] { 2, 4, 5, 5, 6},
            -3
        };

        yield return new object[] {
            new CustomQueue<int>(new int[] { 0, 0, 0, 1}),
            new int[] { 24, 451, -5, 85, 6},
            5
        };
    }

    public static IEnumerable<object[]> NullExceptionCopyToEventTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            null,
            1
        };
    }

    public static IEnumerable<object[]> ArgumentExceptionCopyToEventTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] {31, 34},
            1
        };

        yield return new object[] {
            new CustomQueue<int>(new int[] { 51123, 234, -3251, 23 }),
            new int[] {11, 33, 5},
            0
        };
    }
}
```

Файл ToArrayTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class ToArrayTest
{
    [Theory]
    [MemberData(nameof(ToArrayTestData))]
    public void ToArray_FromIntQueue_SuccessfulConversion(int[] expectedArray)
    {
        CustomQueue<int> cq = new CustomQueue<int>(expectedArray);

        var arr = cq.ToArray();

        Assert.Equal(expectedArray, arr);
    }

    public static IEnumerable<object[]> ToArrayTestData()
```

```
    {
        yield return new object[] { new int[] { 23, 53, 516, -1234, 1234, 51, 6,
73 } };
        yield return new object[] { new int[] { 43, 72, 34, 43 } };
        yield return new object[] { new int[] { -1 } };
        yield return new object[] { new int[] { } };
    }
}
```

Файл ReverseTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class ReverseTest
{
    [Theory]
    [MemberData(nameof(ReverseTestData))]
    public void Reverse_FromIntQueue_SuccessfulReverse(CustomQueue<int>
expectedCq, CustomQueue<int> startingCq)
    {
        CustomQueue<int> cq = startingCq;

        var reversedCq = cq.Reverse();

        Assert.Equal(expectedCq, reversedCq);
    }

    public static IEnumerable<object[]> ReverseTestData()
    {
        yield return new object[]
        {
            new CustomQueue<int>(new int[] { 73, 6, 51, 1234, -1234, 516, 53, 23
}),
            new CustomQueue<int>(new int[] { 23, 53, 516, -1234, 1234, 51, 6, 73
})
        };
        yield return new object[]
        {
            new CustomQueue<int>(new int[] { 341, -2, -2, 51 }),
            new CustomQueue<int>(new int[] { 51, -2, -2, 341})
        };
        yield return new object[]
        {
            new CustomQueue<int>(new int[] { -1 }),
            new CustomQueue<int>(new int[] { -1 })
        };
        yield return new object[]
        {
            new CustomQueue<int>(new int[] { }),
            new CustomQueue<int>(new int[] { })
        };
    }
}
```

Файл CopyToTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class CopyToTest
{
    [Theory]
```

```
    [MemberData(nameof(SuccessfulTestData))]
    public void CopyTo_FromIntQueue_SuccessfulCopy(int[] expectedArray,
CustomQueue<int> startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection;

        cq.CopyTo(array, index);

        Assert.Equal(expectedArray, array);
    }

    [Theory]
    [MemberData(nameof(OutOfRangeIndexTestData))]
    public void CopyTo_FromIntQueue_IndexOutOfRange(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection;

        Assert.Throws<ArgumentOutOfRangeException>(()=>cq.CopyTo(array, index));
    }

    [Theory]
    [MemberData(nameof(ArgumentNullExceptionTestData))]
    public void CopyTo_ToNullArray_NullException(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection;

        Assert.Throws<ArgumentNullException>(()=>cq.CopyTo(array, index));
    }

    [Theory]
    [MemberData(nameof(ArgumentExceptionTestData))]
    public void CopyTo_ToSmallArray_ArgumentException(CustomQueue<int>
startingCollection, int[] array, int index)
    {
        CustomQueue<int> cq = startingCollection;

        Assert.Throws<ArgumentException>(()=>cq.CopyTo(array, index));
    }
    public static IEnumerable<object[]> SuccessfulTestData()
    {
        yield return new object[] {
            new int[] { 2, 4, 232, 441, 55 },
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] { 2, 4, 5, 5, 6},
            2
        };

        yield return new object[] {
            new int[] { 0, 0, 0, 1, 6 },
            new CustomQueue<int>(new int[] { 0, 0, 0, 1}),
            new int[] { 24, 451, -5, 85, 6},
            0
        };

        yield return new object[] {
            new int[] { 24, -3, 124, 34, 2135, 87, 90 },
            new CustomQueue<int>(new int[] { -3, 124, 34, 2135}),
            new int[] { 24, 451, -5, 85, 6, 87, 90},
            1
        };
    }

    public static IEnumerable<object[]> OutOfRangeIndexTestData()
    {
```

```
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] { 2, 4, 5, 5, 6},
            -3
        };

        yield return new object[] {
            new CustomQueue<int>(new int[] { 0, 0, 0, 1}),
            new int[] { 24, 451, -5, 85, 6},
            5
        };
    }

    public static IEnumerable<object[]> ArgumentNullExceptionTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            null,
            1
        };
    }

    public static IEnumerable<object[]> ArgumentExceptionTestData()
    {
        yield return new object[] {
            new CustomQueue<int>(new int[] { 232, 441, 55 }),
            new int[] {31, 34},
            1
        };

        yield return new object[] {
            new CustomQueue<int>(new int[] { 51123, 234, -3251, 23 }),
            new int[] {11, 33, 5},
            0
        };
    }
}
```

Файл ClearTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class ClearTest
{
    [Theory]
    [MemberData(nameof(FilledCollectionTestData))]
    public void Clear_ClearCollection_EmptyCollection(CustomQueue<int>
startingCollection)
    {
        CustomQueue<int> cq = startingCollection;
        cq.Clear();

        Assert.Empty(cq);
    }

    public static IEnumerable<object[]> FilledCollectionTestData()
    {
        yield return new object[] { new CustomQueue<int>(new int[] { 213, 43, -
234, -234, -2 }) };
        yield return new object[] { new CustomQueue<int>() };
    }
}
```

Файл ContsinsTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class ContainsTest
{
    [Theory]
    [MemberData(nameof(ContainsNumberTestData))]
    public void Contains_CheckOnContainingNumber_NumberIsInCollection(int
number, CustomQueue<int> startingCollection)
    {
        CustomQueue<int> cq = startingCollection;
        var contains = cq.Contains(number);

        Assert.True(contains);
    }

    [Theory]
    [MemberData(nameof(NotContainNumberTestData))]
    public void Contains_CheckOnContainingNumber_NumberIsNotInCollection(int
number, CustomQueue<int> startingCollection)
    {
        CustomQueue<int> cq = startingCollection;
        var contains = cq.Contains(number);

        Assert.False(contains);
    }

    public static IEnumerable<object[]> ContainsNumberTestData()
    {
        yield return new object[] { -234, new CustomQueue<int>(new int[] { 213,
43, -234, -234, -2 }) };
        yield return new object[] { 11, new CustomQueue<int>(new int[] { 34, 5,
-2, -87, 90, 11 }) };
        yield return new object[] { 0, new CustomQueue<int>(new int[] { 0, 0, -
2, -3 }) };
    }

    public static IEnumerable<object[]> NotContainNumberTestData()
    {
        yield return new object[] { 5156, new CustomQueue<int>(new int[] { 213,
43, -234, -234, -2 }) };
        yield return new object[] { 12, new CustomQueue<int>(new int[] { 34, 5,
-2, -87, 90, 11 }) };
        yield return new object[] { 5, new CustomQueue<int>(new int[] { 0, 0, -
2, -3 }) };
    }
}
```

Файл DequeueTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class DequeueTest
{
    [Theory]
    [MemberData(nameof(TestData))]
    public void Dequeue_IntElement_SuccessfulDequeue(int expected, int[]
enqueueElements)
    {
        CustomQueue<int> cq = new(enqueueElements);

        var actualElement = cq.Dequeue();
```

```
        Assert.Equal(expected, actualElement);
        Assert.DoesNotContain(expected, cq);
    }

    [Fact]
    public void Dequeue_EmptyCollection_InvalidOperationException()
    {
        CustomQueue<int> cq = new();

        Assert.Throws<InvalidOperationException>(() => cq.Dequeue());
    }

    public static IEnumerable<object[]> TestData()
    {
        yield return new object[] { 2 , new int[] { 2, 4, 5, 5, 6 } };
        yield return new object[] { 132, new int[] { 132, 441, 5412, -55, 612 }
};
        yield return new object[] { -652, new int[] { -652, 524, 551, -7545,
452366 } };
    }
}
```

Файл EnqueueTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class EnqueueTest
{
    [Theory]
    [MemberData(nameof(TestData))]
    public void Enqueue_IntCollection_SuccessfulFilling(CustomQueue<int>
expected, int[] enqueueElements)
    {
        CustomQueue<int> cq = new();

        foreach (var element in enqueueElements)
        {
            cq.Enqueue(element);
        }

        Assert.Equal(expected, cq);
    }

    public static IEnumerable<object[]> TestData()
    {
        yield return new object[] { new CustomQueue<int>(new int[] { 2, 4, 5, 5,
6 }), new int[] { 2, 4, 5, 5, 6 } };
        yield return new object[] { new CustomQueue<int>(new int[] { 132, 441,
5412, 55, 612 }), new int[] { 132, 441, 5412, 55, 612 } };
        yield return new object[] { new CustomQueue<int>(new int[] { 652, 524,
551, 7545, 452366 }), new int[] { 652, 524, 551, 7545, 452366 } };
    }
}
```

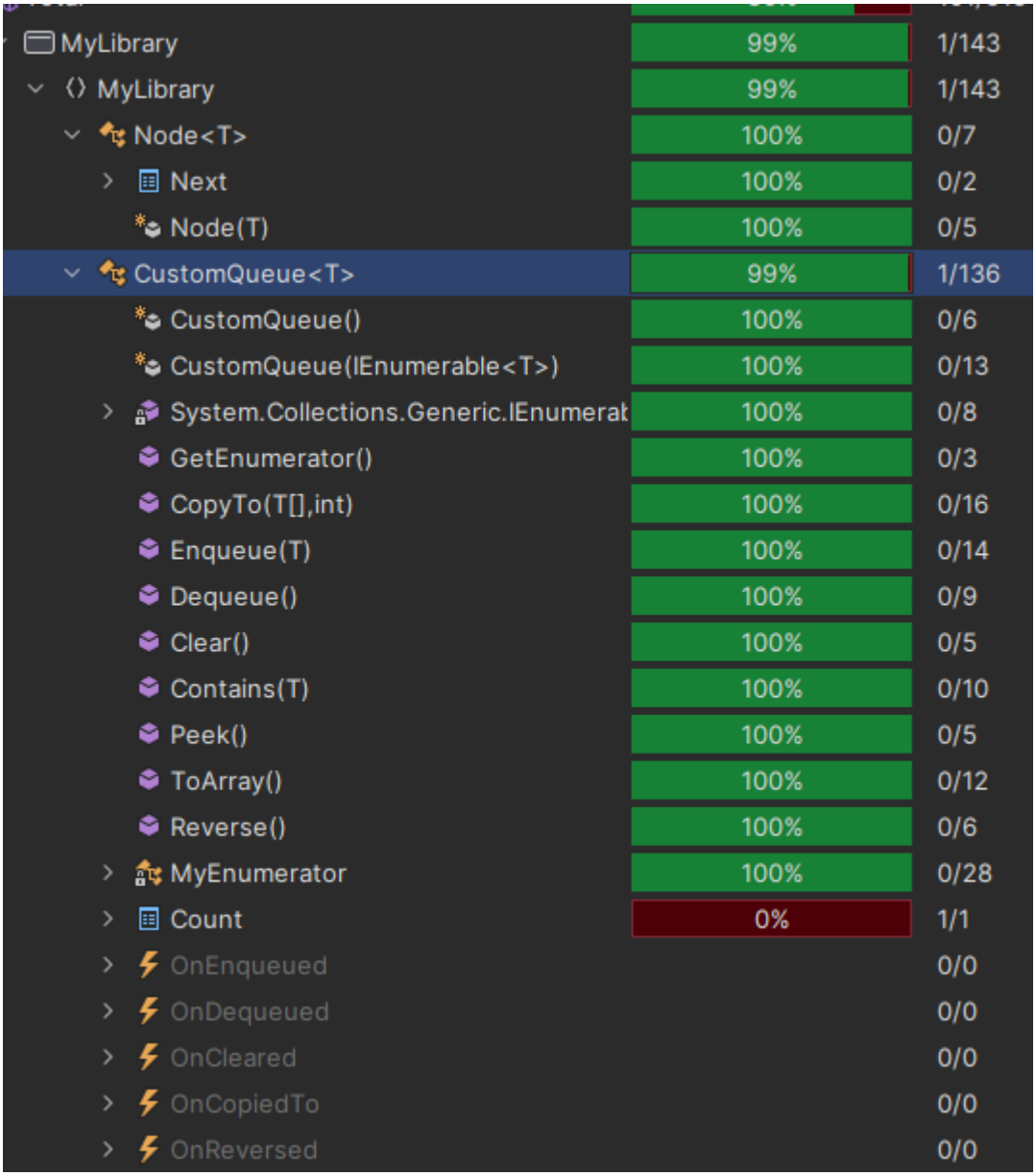
Файл PeekTest.cs:

```
using MyLibrary;
namespace MyLibraryTest;

public class PeekTest
```

```
{

    [Theory]
    [MemberData(nameof(TestData))]
    public void Peek_NotEmptyCollection_FirstElement(int expected, int[]
queueElements)
    {
        CustomQueue<int> cq = new(queueElements);

        var peekedElement = cq.Peek();

        Assert.Equal(expected, peekedElement);
    }

    [Fact]
    public void Peek_EmptyCollection_InvalidOperationException()
    {
        CustomQueue<int> cq = new();

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

    public static IEnumerable<object[]> TestData()
    {
        yield return new object[] { 2, new int[] { 2, 4, 5, 5, 6 } };
        yield return new object[] { 12, new int[] { 12, -44, 35, 25, 16 } };
        yield return new object[] { 123, new int[] { 123, 451, 1, 0, -41235 } };
    }
}
```

2. Результат виконання тестів:

3. Відсоток покриття тестами Колекції:


MyLibrary 99% 1/143
MyLibrary 99% 1/143
Node<T> 100% 0/7
Next 100% 0/2
Node(T) 100% 0/5
CustomQueue<T> 99% 1/136
CustomQueue() 100% 0/6
CustomQueue(IEnumerable<T>) 100% 0/13
System.Collections.Generic.IEnumerat 100% 0/8
GetEnumerator() 100% 0/3
CopyTo(T[],int) 100% 0/16
Enqueue(T) 100% 0/14
Dequeue() 100% 0/9
Clear() 100% 0/5
Contains(T) 100% 0/10
Peek() 100% 0/5
ToArray() 100% 0/12
Reverse() 100% 0/6
MyEnumerator 100% 0/28
Count 0% 1/1
OnEnqueued 0/0
OnDequeued 0/0
OnCleared 0/0
OnCopiedTo 0/0
OnReversed 0/0

Висновок:

Виконуючи дану лабораторну роботу, було розглянуто різні методики написання модульних тестів та застосовано одну з них на практиці. Було написано модульні тести для власно розробленої колекції, використовуючи фреймворк Xunit. Під час тестування функціоналу програми було виявлено та виправлено знайдені недоліки. За допомогою вбудованого в Rider dotCover було перевірено якість покриття коду програми тестами. Покриття коду тестами склало 99%.