Національний технічний університет України «КПІ»

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 гр. ІС-13

Хробатенко Андрій

Викладач:

Бардін В.

2023 рік

**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.

Посилання на GitHub:

<https://github.com/1AnDeX1/Lab2_unit_testing.git>

**Код програми:**

**MyDynamicMassive:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace Lab1

{

internal class MyDynamicMassive<T> : IEnumerable<T>, ICollection<T>, IList<T>

{

private const int DefaultCapacity = 4;

private T[] \_items;

private int \_capacity;

private int \_size;

public event EventHandler<EventInfo<T>>? CollectionCleared;

public event EventHandler<EventInfo<T>>? ItemAdded;

public event EventHandler<EventInfo<T>>? ItemRemoved;

public event EventHandler<EventInfo<T>>? ItemAddedToBeginning;

public event EventHandler<EventInfo<T>>? ItemAddedToEnd;

public event EventHandler<EventInfo<T>>? ItemRemovedAt;

public event EventHandler<EventInfo<T>>? ContainsCheck;

public event EventHandler<EventInfo<T>>? IndexOfCheck;

public MyDynamicMassive(int capacity)

{

if (capacity < 0)

{

throw new ArgumentOutOfRangeException(nameof(capacity));

}

\_capacity = capacity;

\_size = 0;

\_items = capacity is 0

? Array.Empty<T>()

: new T[capacity];

}

public int Count => \_size;

public bool IsReadOnly => false;

public T this[int index]

{

get

{

if (index < 0 || index >= \_size)

{

throw new IndexOutOfRangeException("Index is out of range.");

}

return \_items[index];

}

set

{

if (index < 0 || index >= \_size)

{

throw new IndexOutOfRangeException("Index is out of range.");

}

\_items[index] = value;

}

}

public void Add(T item)

{

if (item == null)

{

throw new ArgumentNullException(nameof(item), "Item cannot be null.");

}

if (\_size >= \_capacity)

{

Resize();

}

\_items[\_size] = item;

\_size++;

ItemAdded?.Invoke(this, new EventInfo<T>("Add", item));

}

public void Clear()

{

\_items = new T[DefaultCapacity];

\_capacity = \_size = 0;

CollectionCleared?.Invoke(this, new EventInfo<T>("Clear", default(T)));

}

public bool Contains(T item)

{

var contains = false;

for (int i = 0; i < \_size; i++)

{

var element = \_items[i];

if (EqualityComparer<T>.Default.Equals(element, item))

{

contains = true;

break;

}

}

ContainsCheck?.Invoke(this, new EventInfo<T>("Contains", item));

return contains;

}

public void CopyTo(T[] array, int arrayIndex)

{

if (array.Length - arrayIndex < \_size)

{

throw new ArgumentException("Dest array is too small");

}

Array.Copy(\_items, 0, array, arrayIndex, \_size);

}

public bool Remove(T item)

{

if (item == null)

{

throw new ArgumentNullException(nameof(item), "Item cannot be null.");

}

var index = Array.IndexOf(\_items, item);

var isRemoved = index != -1;

RemoveAt(index);

ItemRemoved?.Invoke(this, new EventInfo<T>("Remove", item));

return isRemoved;

}

public int IndexOf(T item)

{

var index = Array.IndexOf(\_items, item);

IndexOfCheck?.Invoke(this, new EventInfo<T>("IndexOf", item));

return index;

}

public void Insert(int index, T item)

{

if (index < 0 || index > \_size)

{

throw new ArgumentOutOfRangeException(nameof(index), "Invalid index");

}

if (\_size == \_capacity)

{

Resize();

}

Array.Copy(\_items, index, \_items, index + 1, \_size - index);

\_items[index] = item;

\_size++;

ItemAdded?.Invoke(this, new EventInfo<T>("Insert", item));

}

public void RemoveAt(int index)

{

if (index < 0 || index > \_size)

{

throw new ArgumentOutOfRangeException(nameof(index), "Invalid index.");

}

var removedItem = \_items[index];

\_size--;

Array.Copy(\_items, index + 1, \_items, index, \_size - index);

ItemRemovedAt?.Invoke(this, new EventInfo<T>("RemoveAt", removedItem));

}

public IEnumerator<T> GetEnumerator()

{

return new MyEnumerator(this);

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

private void Resize()

{

var newCapacity = (\_capacity) \* 2;

var tempArray = new T[newCapacity];

Array.Copy(\_items, tempArray, \_size);

\_items = tempArray;

\_capacity = newCapacity;

}

private class MyEnumerator : IEnumerator<T>

{

private MyDynamicMassive<T> \_list;

private int \_cursor;

public T Current

{

get

{

if (\_cursor >= 0 && \_cursor < \_list.\_size)

{

return \_list.\_items[\_cursor];

}

throw new InvalidOperationException();

}

}

object IEnumerator.Current => Current;

public MyEnumerator(MyDynamicMassive<T> list)

{

\_list = list;

\_cursor = -1;

}

public void Dispose()

{

((IEnumerator)this).Reset();

}

public bool MoveNext()

{

if (\_cursor < \_list.Count - 1)

{

\_cursor++;

return true;

}

return false;

}

public void Reset()

{

\_cursor = -1;

}

}

}

}

**EventMethods:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public static class EventMethods<T>

{

public static void CollectionClearedHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Колекція була очищена. Дія: {e.Action}");

}

public static void ItemAddedHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Додано елемент {e.Item}. Дія: {e.Action}");

}

public static void ItemRemovedHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Видалено елемент {e.Item}. Дія: {e.Action}");

}

public static void ItemAddedToBeginningHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Додано елемент {e.Item} на початок. Дія: {e.Action}");

}

public static void ItemAddedToEndHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Додано елемент {e.Item} в кінець. Дія: {e.Action}");

}

public static void ItemRemovedAtHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Видалено елемент {e.Item} за індексом. Дія: {e.Action}");

}

public static void ContainsCheckHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Викликаний метод Contains з параметром {e.Item}. Дія: {e.Action}");

}

public static void IndexOfCheckHandler(object sender, EventInfo<T> e)

{

Console.WriteLine($"Викликаний метод IndexOf з параметром {e.Item}. Дія: {e.Action}");

}

}

}

**EventInfo:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public class EventInfo<T>

{

public string Action { get; set; }

public T Item { get; set; }

public EventInfo(string action, T item)

{

Action = action;

Item = item;

}

}

}

**Program:**

using System.Text;

using MyLibrary;

namespace Lab1

{

class Program

{

public static void Main(string[] args)

{

#region MainCode

Console.OutputEncoding = Encoding.UTF8;

var myCollection = new MyDynamicMassive<int>(4);

myCollection.CollectionCleared += EventMethods<int>.CollectionClearedHandler!;

myCollection.ItemAdded += EventMethods<int>.ItemAddedHandler!;

myCollection.ItemRemoved += EventMethods<int>.ItemRemovedHandler!;

myCollection.ItemAddedToBeginning += EventMethods<int>.ItemAddedToBeginningHandler!;

myCollection.ItemAddedToEnd += EventMethods<int>.ItemAddedToEndHandler!;

myCollection.ItemRemovedAt += EventMethods<int>.ItemRemovedAtHandler!;

myCollection.ContainsCheck += EventMethods<int>.ContainsCheckHandler!;

myCollection.IndexOfCheck += EventMethods<int>.IndexOfCheckHandler!;

myCollection.Add(1);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Add(17);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Add(3);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(0, 0);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(4, 4);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(5, 10);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(6, 6);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(7, 7);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(8, 8);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Insert(9, 9);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

myCollection.Remove(10);

Console.WriteLine("Колекція:");

Println(myCollection);

Console.WriteLine();

int[] destinationArray = new int[myCollection.Count];

myCollection.CopyTo(destinationArray, 0);

Console.WriteLine("Масив призначення після виклику CopyTo:");

foreach (var item in destinationArray)

{

Console.Write($"{item} ");

}

Console.WriteLine();

int valueAtIndex2 = myCollection[2];

Console.WriteLine($"Значення за індексом 2: {valueAtIndex2}");

Console.WriteLine("Зміна значення за індексом 2 на 10");

myCollection[2] = 10;

Console.WriteLine($"Значення за індексом 2 після зміни: {myCollection[2]}");

bool contains5 = myCollection.Contains(10);

Console.WriteLine($"Чи містить колекція значення 10: {contains5}");

int indexOf2 = myCollection.IndexOf(2);

Console.WriteLine($"Індекс елемента 2: {indexOf2}");

Println(myCollection);

int indexOf7 = myCollection.IndexOf(7);

Console.WriteLine($"Індекс елемента 7: {indexOf7}");

Println(myCollection);

Console.WriteLine($"Розмір колекції: {myCollection.Count}");

Console.WriteLine("");

myCollection.RemoveAt(1);

Console.WriteLine("Колекція:");

Println(myCollection);

myCollection.Clear();

Console.WriteLine("Колекція після виконання всіх дій:");

Println(myCollection);

#endregion

#region LibraryCode

Console.WriteLine("Library:");

Console.OutputEncoding = Encoding.UTF8;

var myLibraryCollection = new MyDynamicMassiveLibrary<int>(4);

myLibraryCollection.CollectionCleared += EventMethodsLibrary.CollectionClearedHandler!;

myLibraryCollection.ItemAdded += EventMethodsLibrary.ItemAddedHandler!;

myLibraryCollection.ItemRemoved += EventMethodsLibrary.ItemRemovedHandler!;

myLibraryCollection.ItemAddedToBeginning += EventMethodsLibrary.ItemAddedToBeginningHandler!;

myLibraryCollection.ItemAddedToEnd += EventMethodsLibrary.ItemAddedToEndHandler!;

myLibraryCollection.ItemRemovedAt += EventMethodsLibrary.ItemRemovedAtHandler!;

myLibraryCollection.ContainsCheck += EventMethodsLibrary.ContainsCheckHandler!;

myLibraryCollection.IndexOfCheck += EventMethodsLibrary.IndexOfCheckHandler!;

myLibraryCollection.Add(1);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Add(17);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Add(3);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(0, 0);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(4, 4);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(5, 10);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(6, 6);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(7, 7);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(8, 8);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Insert(9, 9);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

myLibraryCollection.Remove(10);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

Console.WriteLine();

int[] destinationArrayOfLibrary = new int[myLibraryCollection.Count];

myLibraryCollection.CopyTo(destinationArrayOfLibrary, 0);

Console.WriteLine("Масив призначення після виклику CopyTo:");

foreach (var item in destinationArrayOfLibrary)

{

Console.Write($"{item} ");

}

Console.WriteLine();

int valueAtIndex2OfLibrary = myLibraryCollection[2];

Console.WriteLine($"Значення за індексом 2: {valueAtIndex2OfLibrary}");

Console.WriteLine("Зміна значення за індексом 2 на 10");

myLibraryCollection[2] = 10;

Console.WriteLine($"Значення за індексом 2 після зміни: {myLibraryCollection[2]}");

bool contains10OfLibrary = myLibraryCollection.Contains(10);

Console.WriteLine($"Чи містить колекція значення 10: {contains10OfLibrary}");

int indexOf2OfLibrary = myLibraryCollection.IndexOf(2);

Console.WriteLine($"Індекс елемента 2: {indexOf2OfLibrary}");

Println(myLibraryCollection);

int indexOf7OfLibrary = myLibraryCollection.IndexOf(7);

Console.WriteLine($"Індекс елемента 7: {indexOf7OfLibrary}");

Println(myLibraryCollection);

Console.WriteLine($"Розмір колекції: {myLibraryCollection.Count}");

Console.WriteLine("");

myLibraryCollection.RemoveAt(1);

Console.WriteLine("Колекція:");

Println(myLibraryCollection);

myLibraryCollection.Clear();

Console.WriteLine("Колекція після виконання всіх дій:");

Println(myLibraryCollection);

#endregion

Console.ReadLine();

}

private static void Println<T>(MyDynamicMassiveLibrary<T> collection)

{

foreach (var item in collection)

{

Console.Write($"{item} ");

}

Console.WriteLine();

}

private static void Println<T>(MyDynamicMassive<T> collection)

{

foreach (var item in collection)

{

Console.Write($"{item} ");

}

Console.WriteLine();

}

}

}

**DynamicMassiveTests:**

using Lab2\_unit\_testing;

using Microsoft.VisualStudio.TestPlatform.Utilities;

using Moq;

using System.Collections.Generic;

using System.Reflection;

using System.Text;

namespace MyDynamicMassiveTests

{

public class MyDynamicMassiveTests

{

#region MyDynamicMassiveTesting

//Constructor

[Theory]

[InlineData(4)]

[InlineData(6)]

[InlineData(34)]

public void Constructor\_SetNormalCapacity\_ChangingCapacity(int settedcapacity)

{

var capacity = settedcapacity;

var collection = new MyDynamicMassive<int>(capacity);

Assert.Equal(settedcapacity, capacity);

}

[Theory]

[InlineData(-6)]

public void Constructor\_SetWrongCapacity\_ArgumentOutOfRangeException(int settedcapacity)

{

var capacity = settedcapacity;

void action() => new MyDynamicMassive<int>(capacity);

Assert.Throws<ArgumentOutOfRangeException>(action);

}

[Fact]

public void Constructor\_SetZeroCapacity\_InitializesPropertiesCorrectly()

{

var capacity = 0;

var collection = new MyDynamicMassive<int>(capacity);

Assert.Equal(capacity, collection.Count);

Assert.Empty(collection);

}

//Indexer

[Fact]

public void Indexer\_GettingElementsFromArray\_HaveAppropriateElement()

{

var collection = new MyDynamicMassive<int>(3);

collection.Add(1);

collection.Add(2);

collection.Add(3);

var resultofIndex0 = collection[0];

var resultofIndex1 = collection[1];

var resultofIndex2 = collection[2];

Assert.Equal(1, resultofIndex0);

Assert.Equal(2, resultofIndex1);

Assert.Equal(3, resultofIndex2);

}

[Theory]

[InlineData(-1)]

[InlineData(4)]

public void Indexer\_GettingElementsFromArrayWithWrongIndex\_HaveAppropriateElement(int settedIndex)

{

var index = settedIndex;

var collection = new MyDynamicMassive<int>(3);

collection.Add(1);

collection.Add(2);

collection.Add(3);

Action action = () => { var value = collection[settedIndex];};

Assert.Throws<IndexOutOfRangeException>(action);

}

[Fact]

public void Indexer\_SetElementAtIndex1\_ModifiesCorrectValueForIndex1()

{

// Arrange

var collection = new MyDynamicMassive<int>(3);

collection.Add(1);

collection.Add(2);

collection.Add(3);

collection[1] = 42;

Assert.Equal(1, collection[0]);

Assert.Equal(42, collection[1]);

Assert.Equal(3, collection[2]);

}

[Theory]

[InlineData(-1)]

[InlineData(4)]

public void Indexer\_SettingElementsToArrayWithWrongIndex\_HaveAppropriateElement(int settedIndex)

{

var index = settedIndex;

var collection = new MyDynamicMassive<int>(3);

collection.Add(1);

collection.Add(2);

collection.Add(3);

Action action = () => { collection[settedIndex] = 3; };

Assert.Throws<IndexOutOfRangeException>(action);

}

//ReadOnly

[Fact]

public void IsReadOnly\_CheckOnFalse\_ReturnsFalse()

{

var collection = new MyDynamicMassive<int>(3);

var isReadOnly = collection.IsReadOnly;

Assert.False(isReadOnly);

}

//Add method

[Fact]

public void Add\_AddsItemToCollection\_AddingItemAndIncreaceOfCapacity()

{

var collection = new MyDynamicMassive<int>(1);

collection.Add(5);

Assert.Single(collection);

Assert.Equal(5, collection[0]);

}

[Fact]

public void Add\_SetNull\_ArgumentNullException()

{

var collection = new MyDynamicMassive<string>(1);

void action() => collection.Add(null);

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void Add\_ResizesCollection\_ChangingCapacity()

{

var collection = new MyDynamicMassive<int>(1);

collection.Add(1);

collection.Add(2);

collection.Add(3);

collection.Add(4);

collection.Add(5);

collection.Add(6);

Assert.Equal(6, collection.Count);

Assert.Equal(1, collection[0]);

Assert.Equal(2, collection[1]);

Assert.Equal(3, collection[2]);

Assert.Equal(4, collection[3]);

Assert.Equal(5, collection[4]);

Assert.Equal(6, collection[5]);

}

//Clear method

[Fact]

public void Clear\_SettingFiveItems\_ClearingMassive()

{

var collection = new MyDynamicMassive<int>(5)

{

1,2,3,4,5

};

collection.Clear();

Assert.Empty(collection);

}

[Fact]

public void Clear\_SettingNull\_NullReferenceException()

{

var collection = new MyDynamicMassive<string>(10);

typeof(MyDynamicMassive<string>).GetField("\_items", BindingFlags.Instance | BindingFlags.NonPublic)?.SetValue(collection, null);

void action() => collection.Clear();

Assert.Throws<NullReferenceException>(action);

}

//Contains method

[Theory]

[InlineData(1)]

[InlineData(2)]

[InlineData(3)]

public void Contains\_SearchingOfExistingNumber\_True(int a)

{

var collection = new MyDynamicMassive<int>(7)

{

1,2, 3, 4, 5, 6, 7,

};

bool result = collection.Contains(a);

Assert.True(result);

}

[Theory]

[InlineData(34)]

[InlineData(8)]

[InlineData(9)]

public void Contains\_SearchingOfUnexistingNumber\_False(int a)

{

var collection = new MyDynamicMassive<int>(7)

{

1,2, 3, 4, 5, 6, 7,

};

bool result = collection.Contains(a);

Assert.False(result);

}

[Fact]

public void Contains\_SetNull\_ArgumentNullException()

{

var collection = new MyDynamicMassive<string>(7);

void action() => collection.Contains(null);

Assert.Throws<ArgumentNullException>(action);

}

//CopyTo

[Fact]

public void CopyTo\_SetArrayWithEqualLength\_CopiedArray()

{

var mainCollection = new MyDynamicMassive<int>(5)

{

1,2, 3, 4, 5

};

int[] coppiedCollection = new int[5];

mainCollection.CopyTo(coppiedCollection, 0);

Assert.NotEmpty(coppiedCollection);

Assert.Equal(mainCollection, coppiedCollection);

}

[Fact]

public void CopyTo\_SetArrayWithMoreLengthFromTheSecondElement\_CopiedArray()

{

var collection = new MyDynamicMassive<int>(3)

{

1, 2, 3

};

var destinationArray = new int[5];

collection.CopyTo(destinationArray, 1);

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

}

[Fact]

public void CopyTo\_SetArrayWithLessLength\_ArgumentException()

{

var mainCollection = new MyDynamicMassive<int>(5)

{

1,2, 3, 4, 5

};

int[] coppiedCollection = new int[4];

void action() => mainCollection.CopyTo(coppiedCollection, 0);

Assert.Throws<ArgumentException>(action);

}

[Fact]

public void CopyTo\_SetNullArray\_ArgumentNullException()

{

var mainCollection = new MyDynamicMassive<int>(5)

{

1,2, 3, 4, 5

};

int[]? coppiedCollection = null;

void action() => mainCollection.CopyTo(coppiedCollection, 0);

Assert.Throws<ArgumentNullException>(action);

}

//RemoveAt

[Fact]

public void RemoveAt\_RemovingAtIndex0\_RemovedItemAtIndex0()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

collection.RemoveAt(0);

Assert.Equal(new int[] { 2, 3, 4, 5 }, collection);

}

[Fact]

public void RemoveAt\_RemovingAtIndex2\_RemovedItemAtIndex2()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

collection.RemoveAt(2);

Assert.Equal(new int[]{1, 2, 4, 5}, collection);

}

[Fact]

public void RemoveAt\_RemovingAtIndex0\_RemovedItem2()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

var size = collection.Count;

collection.RemoveAt(0);

Assert.Equal(size - 1 , collection.Count);

Assert.Equal(new int[] { 2, 3, 4, 5 }, collection);

}

[Theory]

[InlineData(-5)]

[InlineData(10)]

[InlineData(5)]

public void RemoveAt\_SetWrongIndex\_ArgumentOutOfRangeException(int index)

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

void action() => collection.RemoveAt(index);

Assert.Throws<ArgumentOutOfRangeException>(action);

}

//Remove

[Fact]

public void Remove\_RemovingItem1\_RemovedItem1()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

collection.Remove(1);

Assert.Equal(new int[] { 2, 3, 4, 5 }, collection);

}

[Fact]

public void Remove\_RemovingItem4\_RemovedItem4()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

collection.Remove(4);

Assert.Equal(new int[] {1, 2, 3, 5 }, collection);

}

[Fact]

public void Remove\_SetNull\_ArgumentNullException()

{

var collection = new MyDynamicMassive<string>(5) {"1", "2", "3", "4", "5" };

void action() => collection.Remove(null);

Assert.Throws<ArgumentNullException>(action);

}

//IndexOf

[Fact]

public void IndexOf\_FindIndexOfNumber2\_IndexOfNumber2()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

var index = collection.IndexOf(2);

Assert.Equal(1, index);

}

[Fact]

public void IndexOf\_SetUnExistedItem\_OutputMinus1()

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

var index = collection.IndexOf(32);

Assert.Equal(-1, index);

}

[Fact]

public void IndexOf\_SetNullItem\_ArgumentNullException()

{

var collection = new MyDynamicMassive<string>(5) {"1", "2", "3", "4", "5"};

void action() => collection.IndexOf(null);

Assert.Throws<ArgumentNullException>(action);

}

//Insert

[Theory]

[InlineData(0, 5)]

[InlineData(1, 17)]

[InlineData(2, 15)]

[InlineData(3, 7)]

[InlineData(5, 32)]

public void Insert\_InsertingIntoThirdPosition\_ChanginItemIntoThirdPosition(int index, int item)

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

collection.Insert(index, item);

Assert.Equal(6, collection.Count);

Assert.Equal(item, collection[index]);

}

[Fact]

public void Insert\_SetNullItem\_ArgumentNullException()

{

var collection = new MyDynamicMassive<string>(5) {"1", "2", "3", "4", "5"};

void action() => collection.Insert(1, null);

Assert.Throws<ArgumentNullException>(action);

}

[Theory]

[InlineData(-5, 17)]

[InlineData(23, 15)]

public void Insert\_SetWrongIndex\_ArgumentOutOfRangeException(int index, int item)

{

var collection = new MyDynamicMassive<int>(5) { 1, 2, 3, 4, 5 };

void action() => collection.Insert(index, item);

Assert.Throws<ArgumentOutOfRangeException>(action);

}

#endregion

#region Enumerator class

//Enumerator

[Fact]

public void Enumerator\_GettingNumber\_AppropriateNumber()

{

var collection = new MyDynamicMassive<int>(3) { 1, 2, 3 };

var enumerator = collection.GetEnumerator();

enumerator.MoveNext();

var result = enumerator.Current;

Assert.Equal(1, result);

}

[Fact]

public void Enumerator\_Current\_ShouldThrowInvalidOperationExceptionIfCursorIsInvalid()

{

MyDynamicMassive<int> collection = new MyDynamicMassive<int>(3);

var enumerator = collection.GetEnumerator();

enumerator.MoveNext();

Assert.Throws<InvalidOperationException>(() =>

{

var current = enumerator.Current;

});

}

//Constructor

[Fact]

public void MyEnumerator\_Constructor\_NullList\_ThrowsArgumentNullException()

{

MyDynamicMassive<int> list = null;

void action() => new MyDynamicMassive<int>.MyEnumerator(list);

Assert.Throws<ArgumentNullException>(action);

}

//MoveNext

[Fact]

public void MoveNext\_SettingCursorIsBeforeTheStart\_ReturnsTrueOnFirstCall()

{

var collection = new MyDynamicMassive<int>(3){1,2,3};

var enumerator = collection.GetEnumerator();

bool moveNextResult = enumerator.MoveNext();

Assert.True(moveNextResult);

}

[Fact]

public void MoveNext\_SettingCursorIsAtEnd\_ReturnsFalse()

{

var collection = new MyDynamicMassive<int>(3) { 1, 2, 3 };

var enumerator = collection.GetEnumerator();

while (enumerator.MoveNext()) { }

var result = enumerator.MoveNext();

Assert.False(result);

}

//[Fact]

//public void MoveNext\_SettingCursorIsAtEnd\_Retue()

//{

// MyDynamicMassive<int>.MyEnumerator list = null;

// void action() => list?.MoveNext();

// Assert.Throws<ArgumentNullException>(action);

//}

#endregion

#region EventMethodsTesting

[Fact]

public void CollectionClearedHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "Clear",

Item = 0 // Встановлюємо значення за замовчуванням для int

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.CollectionClearedHandler(null, eventInfo));

// Assert

Assert.Contains("Колекція була очищена. Дія: Clear", capturedOutput);

}

[Fact]

public void ItemAddedHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "Add",

Item = 5

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ItemAddedHandler(null, eventInfo));

// Assert

Assert.Contains("Додано елемент 5. Дія: Add", capturedOutput);

}

[Fact]

public void ItemRemovedHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "Remove",

Item = 10

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ItemRemovedHandler(null, eventInfo));

// Assert

Assert.Contains("Видалено елемент 10. Дія: Remove", capturedOutput);

}

[Fact]

public void ItemAddedToBeginningHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "AddToBeginning",

Item = 3

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ItemAddedToBeginningHandler(null, eventInfo));

// Assert

Assert.Contains("Додано елемент 3 на початок. Дія: AddToBeginning", capturedOutput);

}

[Fact]

public void ItemAddedToEndHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "AddToEnd",

Item = 8

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ItemAddedToEndHandler(null, eventInfo));

// Assert

Assert.Contains("Додано елемент 8 в кінець. Дія: AddToEnd", capturedOutput);

}

[Fact]

public void ItemRemovedAtHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "RemoveAt",

Item = 2

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ItemRemovedAtHandler(null, eventInfo));

// Assert

Assert.Contains("Видалено елемент 2 за індексом. Дія: RemoveAt", capturedOutput);

}

[Fact]

public void ContainsCheckHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "ContainsCheck",

Item = 7

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.ContainsCheckHandler(null, eventInfo));

// Assert

Assert.Contains("Викликаний метод Contains з параметром 7. Дія: ContainsCheck", capturedOutput);

}

[Fact]

public void IndexOfCheckHandler\_ShouldPrintMessageToConsole()

{

// Arrange

var eventInfo = new EventInfo<int>

{

Action = "IndexOfCheck",

Item = 4

};

// Act

var capturedOutput = CaptureOutput(() => EventMethods<int>.IndexOfCheckHandler(null, eventInfo));

// Assert

Assert.Contains("Викликаний метод IndexOf з параметром 4. Дія: IndexOfCheck", capturedOutput);

}

private string CaptureOutput(Action action)

{

var output = new StringBuilder();

var originalOutput = Console.Out;

Console.SetOut(new StringWriter(output));

try

{

action.Invoke();

return output.ToString().Trim();

}

finally

{

Console.SetOut(originalOutput);

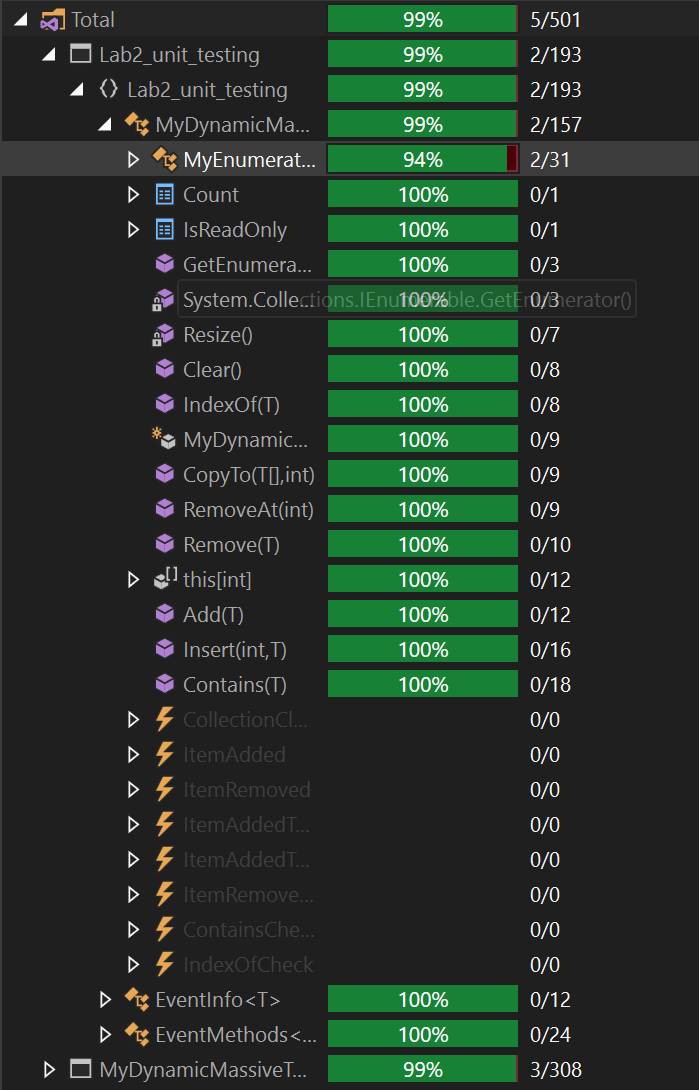
}

}

#endregion

}

}

****

**Висновок:**

Я навчився створювати модульні тести для вихідного коду розроблювального програмного забезпечення.