**Міністерство освіти і науки України**
**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра обчислювальної техніки**

# Лабораторна робота №2

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

Виконав:

студент групи ІМ-12
Кривенок Максим Геннадійович
варіант: 6

Перевірила:

Крамар Ю. М.

Київ 2023

# Завдання

1. Додати до проекту власної узагальненої колекції (застосувати виконану лабораторну роботу №1) 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 виконання роботи

1. Додав до проекту власної узагальненої колекції (застосував виконану лабораторну роботу №1) проект модульних тестів, використовуючи фреймворк MSTest. Для цього в Visual Studio відкрив рішення з проектом колекції, вибрав меню File -> Add -> New Project, обрав тип проекту MSTest Test Project (.NET Core), дав йому назву **DotNet_Lab1-2.Core.Tests** та натиснув OK. Потім додав посилання на проект колекції в проекті тестів, вибравши меню Project -> Add Reference, обравши проект Collection та натиснувши OK.
2. Розробив модульні тести для функціоналу колекції. Для цього в файлі **MyDictionaryTests.cs** в проекті тестів написав код, який наведено нижче. Для тестування в фреймворці MSTest я використовував наступні атрибути та методи:
    1. Атрибут [TestClass] для позначення класу, що містить тести.
    2. Атрибут [TestMethod] для позначення методу, що є тестом.
    3. Клас Assert для перевірки очікуваних умов та значень.
    4. Методи Assert.AreEqual, Assert.IsTrue, Assert.IsFalse та інші для порівняння фактичних та очікуваних результатів.
    5. Клас CollectionAssert для перевірки умов, пов’язаних з колекціями.
    6. Метод CollectionAssert.AreEqual рівності елементів колекцій.
3. Дослідив ступінь покриття модульними тестами вихідного коду колекції, використовуючи засіб **Fine Code Coverage**. Для цього в Visual Studio встановив розширення Fine Code Coverage з меню Extensions -> Manage Extensions, перезапустив Visual Studio, відкрив рішення з проєктом колекції та проєктом тестів, вибрав меню Test -> Run All Tests, дочекався результатів

тестів та переглянув вікно Fine Code Coverage, де було показано, ступінь покриття класу тестами, а вже в коді класу я зміг побачити які рядки коду були виконані тестами, а які ні. За допомогою цього засобу я виявив, що мої тести покривають 96.2% коду колекції, що частково свідчить про їхню якість та достатність.

# Код

- **MyDictionaryTests.cs**

```
using System.Collections;

namespace DotNet_Lab.Core.Tests
{
    [TestClass]
    public class MyDictionaryTests
    {
        #region Add
        [TestMethod]
        public void Add_KeyValue_Count()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";
            int value = 1;

            myDictionary.Add(key, value);

            int count = myDictionary.Count;
            Assert.AreEqual(1, count);
        }

        [TestMethod]
        public void Add_KeyValuePair()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";
            int value = 1;
            KeyValuePair<string, int> kvp = new(key, value);

            myDictionary.Add(kvp);

            int count = myDictionary.Count;
            Assert.AreEqual(1, count);
        }
        #endregion

        #region IndexerGetter
        [TestMethod]
        public void IndexerGetter_ValidValue_ReturnValidValue()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary = [new(key, value)];

            int resultValue = myDictionary[key];

            Assert.AreEqual(value, resultValue);
```

```
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void IndexerGetter_NullKey_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            string? key = null;

            int resultValue = myDictionary[key];
        }

        [TestMethod]
        [ExpectedException(typeof(KeyNotFoundException))]
        public void IndexerGetter_NonExistingKey_ShouldThrowKeyNotFoundException()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "NonExistingKey";

            int resultValue = myDictionary[key];
        }
        #endregion

        #region IndexerSetter
        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void IndexerSetter_NullKey_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            string? key = null;
            int value = 1;

            myDictionary[key] = value;
        }

        [TestMethod]
        public void IndexerSetter_NonExistingKey_AddNewElement()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";
            int value = 1;

            myDictionary[key] = value;

            int resultValue = myDictionary[key];
            int count = myDictionary.Count;
            Assert.AreEqual(value, resultValue, "Значення елементу відрізнається від
очікуваного");
            Assert.AreEqual(1, count, "Кількість елементів в словнику відрізняється від
очікуваної");
        }

        [TestMethod]
        public void IndexerSetter_ExistingKey_ChangeExistingElement()
        {
            string key = "key";
            int value = 1;
            int newValue = 1;
            MyDictionary<string, int> myDictionary = [new(key, value)];

            myDictionary[key] = newValue;
```

```
            int resultValue = myDictionary[key];
            int count = myDictionary.Count;
            Assert.AreEqual(newValue, resultValue, "Значення елементу відрізнається від
очікуваного");
            Assert.AreEqual(1, count, "Кількість елементів в словнику відрізняється від
очікуваної");
        }
        #endregion

        #region Insert
        [DataTestMethod]
        [DataRow(0)]
        [DataRow(1)]
        [DataRow(2)]
        public void Insert_AtValidIndex_NewElementAtPosition(int index)
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary =
            [
                new($"{key}1", value + 1),
                new($"{key}2", value + 2),
                new($"{key}3", value + 3),
            ];

            myDictionary.Insert(key, value, index);

            int resultValue = myDictionary[key];
            int resultValueAtIndex = GetValue(myDictionary);
            int count = myDictionary.Count;
            Assert.AreEqual(value, resultValue, "Значення елементу відрізнається від
очікуваного");
            Assert.AreEqual(value, resultValueAtIndex, "Значення елементу в заданій
позиції відрізнається від очікуваного");
            Assert.AreEqual(4, count, "Кількість елементів в словнику відрізняється від
очікуваної");

            int GetValue(MyDictionary<string, int> dictionary)
            {
                int i = 0;
                foreach (var (_, v) in dictionary)
                {
                    if(i++ == index)
                    {
                        return v;
                    }
                }

                return -1;
            }
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void Insert_NullKey_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            string? key = null;
            int value = 1;
```

```
            myDictionary.Insert(key, value, 0);
        }

        [DataTestMethod]
        [DataRow(-1)]
        [DataRow(3)]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]
        public void Insert_AtInvalidIndex_ShouldThrowArgumentOutOfRangeException(int
invalidIndex)
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";
            int value = 1;

            myDictionary.Insert(key, value, invalidIndex);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void Insert_ExistingKey_ShouldThrowArgumentException()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary = [new(key, value)];

            myDictionary.Insert(key, value, 0);
        }
        #endregion

        #region Clear
        [TestMethod]
        public void Clear_AllElementsRemoved()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary = [new(key, value)];

            myDictionary.Clear();

            int count = myDictionary.Count;
            Assert.AreEqual(0, count, "Кількість елементів в словнику відрізняється від
очікуваної");
            Assert.ThrowsException<KeyNotFoundException>(() => myDictionary[key]);
        }
        #endregion

        #region Contains
        [TestMethod]
        public void Contains_ExistingElement_True()
        {
            string key = "key";
            int value = 1;
            KeyValuePair<string, int> kvp = new(key, value);
            MyDictionary<string, int> myDictionary = [kvp];

            var isContains = myDictionary.Contains(kvp);

            Assert.IsTrue(isContains);
        }

        [TestMethod]
```

```
public void Contains_NonExistingElement_False()
{
    string key = "key";
    int value = 1;
    KeyValuePair<string, int> kvp = new(key, value);
    MyDictionary<string, int> myDictionary = [];

    var isContains = myDictionary.Contains(kvp);

    Assert.IsFalse(isContains);
}
#endregion

#region ContainsKey
[TestMethod]
public void ContainsKey_ExistingElement_True()
{
    string key = "key";
    int value = 1;
    MyDictionary<string, int> myDictionary = [new(key, value)];

    var isContains = myDictionary.ContainsKey(key);

    Assert.IsTrue(isContains);
}

[TestMethod]
public void ContainsKey_NonExistingKey_False()
{
    MyDictionary<string, int> myDictionary = [];
    string key = "key";

    var isContains = myDictionary.ContainsKey(key);

    Assert.IsFalse(isContains);
}

[TestMethod]
[ExpectedException(typeof(ArgumentNullException))]
public void ContainsKey_NullKey_ShouldThrowArgumentNullException()
{
    MyDictionary<string, int> myDictionary = [];
    string? key = null;

    myDictionary.ContainsKey(key);
}
#endregion

#region CopyTo
[TestMethod]
public void CopyTo_AtValidIndex_NewElementAtPosition()
{
    string key = "key";
    int value = 1;
    MyDictionary<string, int> myDictionary =
    [
        new($"{key}1", value),
        new($"{key}2", value),
    ];
    var array = new KeyValuePair<string, int>[2];
```

```
            myDictionary.CopyTo(array, 0);

            CollectionAssert.AreEqual(myDictionary.ToList(), array);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void CopyTo_NullArray_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            KeyValuePair<string, int>[]? array = null;

            myDictionary.CopyTo(array, 0);
        }

        [DataTestMethod]
        [DataRow(-1)]
        [DataRow(3)]
        [ExpectedException(typeof(ArgumentOutOfRangeException))]
        public void CopyTo_ArrayInvalidIndex_ShouldThrowArgumentOutOfRangeException(int
invalidIndex)
        {
            MyDictionary<string, int> myDictionary = [];
            var array = new KeyValuePair<string, int>[2];

            myDictionary.CopyTo(array, invalidIndex);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void CopyTo_InvalidArrayLength_ShouldThrowArgumentException()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary =
            [
                new($"{key}1", value),
                new($"{key}2", value),
            ];
            var array = new KeyValuePair<string, int>[1];

            myDictionary.CopyTo(array, 0);
        }
        #endregion

        #region Remove
        [TestMethod]
        public void Remove_ExistingKey_True()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary =
            [
                new($"{key}Beggining", value),
                new(key, value),
                new($"{key}End", value)
            ];

            var isRemoved = myDictionary.Remove(key);

            int count = myDictionary.Count;
```

```
            Assert.IsTrue(isRemoved);
            Assert.AreEqual(2, count, "Кількість елементів в словнику відрізняється від
очікуваної");
            Assert.ThrowsException<KeyNotFoundException>(() => myDictionary[key]);

        }

        [TestMethod]
        public void Remove_NonExistingKey_False()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";

            var isRemoved = myDictionary.Remove(key);

            int count = myDictionary.Count;
            Assert.IsFalse(isRemoved);
            Assert.AreEqual(0, count, "Кількість елементів в словнику відрізняється від
очікуваної");
            Assert.ThrowsException<KeyNotFoundException>(() => myDictionary[key]);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void Remove_NullKey_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            string? key = null;

            myDictionary.Remove(key);
        }

        [TestMethod]
        public void Remove_ExistingKVPair_True()
        {
            string key = "key";
            int value = 1;
            KeyValuePair<string, int> kvp = new(key, value);
            MyDictionary<string, int> myDictionary = [kvp];

            var isRemoved = myDictionary.Remove(kvp);

            int count = myDictionary.Count;
            Assert.IsTrue(isRemoved);
            Assert.AreEqual(0, count, "Кількість елементів в словнику відрізняється від
очікуваної");
            Assert.ThrowsException<KeyNotFoundException>(() => myDictionary[key]);
        }

        [TestMethod]
        public void Remove_NonExistingKVPair_False()
        {
            string key = "key";
            int value = 1;
            KeyValuePair<string, int> kvp = new(key, value);
            MyDictionary<string, int> myDictionary = [];

            var isRemoved = myDictionary.Remove(kvp);

            int count = myDictionary.Count;
            Assert.IsFalse(isRemoved);
```

```
            Assert.AreEqual(0, count, "Кількість елементів в словнику відрізняється від
очікуваної");
            Assert.ThrowsException<KeyNotFoundException>(() => myDictionary[key]);
        }
        #endregion

        #region TryGetValue
        [TestMethod]
        public void TryGetValue_ExistingKey_True()
        {
            string key = "key";
            int value = 1;
            MyDictionary<string, int> myDictionary = [new(key, value)];

            var isContains = myDictionary.TryGetValue(key, out int resultValue);

            Assert.IsTrue(isContains);
            Assert.AreEqual(value, resultValue);

        }

        [TestMethod]
        public void TryGetValue_NonExistingKey_False()
        {
            MyDictionary<string, int> myDictionary = [];
            string key = "key";

            var isContains = myDictionary.TryGetValue(key, out int resultValue);

            Assert.IsFalse(isContains);
            Assert.AreEqual(resultValue, default);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        public void TryGetValue_NullKey_ShouldThrowArgumentNullException()
        {
            MyDictionary<string, int> myDictionary = [];
            string? key = null;

            myDictionary.TryGetValue(key, out int _);
        }
        #endregion

        #region GetEnumerator
        [TestMethod]
        public void GetEnumerator_ExistingKey_True()
        {
            KeyValuePair<string, int>[] expectedValues = [new("1", 1), new("2", 2),
new("3", 3)];
            MyDictionary<string, int> myDictionary = [.. expectedValues];

            IEnumerator e1 = expectedValues.GetEnumerator();
            IEnumerator e2 = myDictionary.GetEnumerator();
            Assert.AreEqual(expectedValues.Length, myDictionary.Count);
            while (e2.MoveNext() && e1.MoveNext())
            {
                Assert.AreEqual(e1.Current, e2.Current);
            }
        }
        #endregion
```

```
        #region Reverse
        [TestMethod]
        public void Reverse_ExistingKey_True()
        {
            KeyValuePair<string, int>[] expectedValues = [new("1", 1), new("2", 2),
new("3", 3)];
            MyDictionary<string, int> myDictionary = [.. expectedValues];

            IEnumerator e1 = expectedValues.Reverse().GetEnumerator();
            IEnumerator e2 = myDictionary.Reverse().GetEnumerator();
            Assert.AreEqual(expectedValues.Length, myDictionary.Count);
            while (e2.MoveNext() && e1.MoveNext())
            {
                Assert.AreEqual(e1.Current, e2.Current);
            }
        }
        #endregion

        #region Events
        [TestMethod]
        public void Clear_CollectionCleared()
        {
            int eventFiriedCount = 0;
            MyDictionary<string, int> myDictionary = [];
            myDictionary.CollectionCleared += (_) => eventFiriedCount++;

            myDictionary.Clear();

            Assert.AreEqual(1, eventFiriedCount);
        }

        [TestMethod]
        public void CopyTo_CollectionCopied()
        {
            string key = "key";
            int value = 1;
            int eventFiriedCount = 0;
            MyDictionary<string, int> myDictionary = [];
            myDictionary.CollectionCopied += (_, _) => eventFiriedCount++;

            myDictionary.CopyTo([new(key, value)], 0);

            Assert.AreEqual(1, eventFiriedCount);
        }

        [TestMethod]
        public void Add_ElementAdded()
        {
            string key = "key";
            int value = 1;
            int eventFiriedCount = 0;
            MyDictionary<string, int> myDictionary = [];
            myDictionary.ElementAdded += (_, _) => eventFiriedCount++;

            myDictionary.Insert(key, value, 0);

            Assert.AreEqual(1, eventFiriedCount);
        }

        [TestMethod]
```

```
        public void IndexerSetter_ElementChanged()
        {
            string key = "key";
            int value = 1;
            int eventFiriedCount = 0;
            MyDictionary<string, int> myDictionary = [new(key, value)];
            myDictionary.ElementChanged += (_, _, _) => eventFiriedCount++;

            myDictionary[key] = value + 1;

            Assert.AreEqual(1, eventFiriedCount);
        }

        [TestMethod]
        public void Remove_ElementRemoved()
        {
            string key = "key";
            int value = 1;
            int eventFiriedCount = 0;
            MyDictionary<string, int> myDictionary = [new(key, value)];
            myDictionary.ElementRemoved += (_, _) => eventFiriedCount++;

            myDictionary.Remove(key);

            Assert.AreEqual(1, eventFiriedCount);
        }
        #endregion
    }
}
```

# Висновки

У ході виконання лабораторної роботи я навчився створювати модульні тести для вихідного коду розроблювального програмного забезпечення, використовуючи фреймворк MSTest та засіб Fine Code Coverage. Я додав проект модульних тестів до свого проекту колекції, розробив тести для функціоналу колекції та дослідив ступінь покриття модульними тестами вихідного коду колекції. Я виявив, що мої тести покривають 96.2% коду колекції, що свідчить про їхню якість та достатність. Я зрозумів, що модульне тестування є важливим етапом розробки програмного забезпечення, оскільки воно дозволяє перевірити правильність роботи окремих частин коду, виявити та виправити помилки, підвищити надійність та якість програмного продукту.