**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ»**

**Зображення, що містить будівля, ескіз, панорама, чорно-білий

Автоматично згенерований опис**

**Кафедра інформаційних систем та технологій**

Лабораторна робота №2

з дисципліни «Розробка програмного забезпечення на платформі .Net»

на тему:

«Модульне тестування. Ознайомлення з засобами та практиками

модульного тестування»

|  |  |  |
| --- | --- | --- |
| Викладачк:  Бардін В. |  | Виконав:                                               Студент групи ІС-11  Петраков Назар |

Київ – 2023

**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.

Програмний код:

AddingTests.cs

namespace MyCollection.Tests

{

public class AddingTests

{

[Fact]

public void AddItem\_NullKey\_ArgumentNullException()

{

MyDictionary<object, object> myDictionary = new MyDictionary<object, object>();

var action = () => myDictionary.Add(null!, 2);

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void AddItem\_KeyAlreadyExist\_ArgumentException()

{

MyDictionary<object, object> myDictionary = new MyDictionary<object, object>();

var action = () =>

{

myDictionary.Add(4, "four");

myDictionary.Add(4, "five");

};

Assert.Throws<ArgumentException>(action);

}

[Theory]

[MemberData(nameof(TestDataGenerator.GetNumbersFromDataGenerator), MemberType = typeof(TestDataGenerator))]

public void AddItems\_ValidData\_CollectionWithItems(KeyValuePair<object, object> kvp1, KeyValuePair<object, object> kvp2)

{

MyDictionary<object, object> myDictionary = new MyDictionary<object, object>();

myDictionary.Add(kvp1);

myDictionary.Add(kvp2);

Assert.Equal(kvp1.Value, myDictionary[kvp1.Key]);

Assert.True(myDictionary.ContainsKey(kvp2.Key));

}

}

}

ContainsTests.cs

namespace MyCollection.Tests

{

public class ContainsTests

{

[Fact]

public void ContainsKeyValue\_ExistingPair\_ReturnsTrue()

{

var myDictionary = new MyDictionary<int, string>();

myDictionary.Add(1, "One");

bool result = myDictionary.Contains(new KeyValuePair<int, string>(1, "One"));

Assert.True(result);

}

[Fact]

public void ContainsKeyValue\_NotExistPair\_ReturnsFalse()

{

var myDictionary = new MyDictionary<int, string>();

bool result = myDictionary.Contains(new KeyValuePair<int, string>(1, "One"));

Assert.False(result);

}

[Fact]

public void Contains\_NullKey\_ThrowsArgumentNullException()

{

var myDictionary = new MyDictionary<object, string>();

Action action = () => myDictionary.Contains(new KeyValuePair<object, string>(null!, "One"));

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void ContainsKey\_ExistingKey\_ReturnsTrue()

{

var myDictionary = new MyDictionary<int, string>();

myDictionary.Add(1, "One");

bool result = myDictionary.ContainsKey(1);

Assert.True(result);

}

[Fact]

public void ContainsKey\_NonExistentKey\_ReturnsFalse()

{

var myDictionary = new MyDictionary<int, string>();

bool result = myDictionary.ContainsKey(1);

Assert.False(result);

}

[Fact]

public void ContainsKey\_NullKey\_ThrowsArgumentNullException()

{

var myDictionary = new MyDictionary<object, string>();

Action action = () => myDictionary.ContainsKey(null!);

Assert.Throws<ArgumentNullException>(action);

}

}

}

CopyToTests.cs

namespace MyCollection.Tests

{

public class CopyToTests

{

[Fact]

public void CopyTo\_CopiesElementsToDestinationArray()

{

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

{ 2, "Two" },

{ 3, "Three" }

};

var array = new KeyValuePair<int, string>[3];

myDictionary.CopyTo(array, 0);

Assert.Equal(new KeyValuePair<int, string>(1, "One"), array[0]);

Assert.Equal(new KeyValuePair<int, string>(2, "Two"), array[1]);

Assert.Equal(new KeyValuePair<int, string>(3, "Three"), array[2]);

}

[Fact]

public void CopyTo\_NullArray\_ArgumentNullException()

{

var myDictionary = new MyDictionary<int, string>();

var array = (KeyValuePair<int, string>[])null!;

Action action = () => myDictionary.CopyTo(array, 0);

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void CopyTo\_NegativeArrayIndex\_ArgumentOutOfRangeException()

{

var myDictionary = new MyDictionary<int, string>();

var array = new KeyValuePair<int, string>[3];

Action action = () => myDictionary.CopyTo(array, -1);

Assert.Throws<ArgumentOutOfRangeException>(action);

}

[Fact]

public void CopyTo\_ArrayIndexOutOfRange\_ArgumentOutOfRangeException()

{

var myDictionary = new MyDictionary<int, string>();

var array = new KeyValuePair<int, string>[3];

Action action = () => myDictionary.CopyTo(array, 5);

Assert.Throws<ArgumentOutOfRangeException>(action);

}

[Fact]

public void CopyTo\_ArrayTooSmall\_ArgumentException()

{

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

{ 2, "Two" }

};

var array = new KeyValuePair<int, string>[1];

Action action = () => myDictionary.CopyTo(array, 0);

Assert.Throws<ArgumentException>(action);

}

}

}

CreationTests.cs

namespace MyCollection.Tests

{

public class CreationTests

{

[Fact]

public void CreateMyDictionary\_WithoutParams\_EmptyDictionary()

{

MyDictionary<object, object> myDictionary;

myDictionary = new MyDictionary<object, object>();

Assert.Empty(myDictionary);

}

[Fact]

public void CreateMyDictionary\_CapacityLessThanZero\_ArgumentOutOfRangeException()

{

MyDictionary<object, object> myDictionary;

var action = () => { myDictionary = new MyDictionary<object, object>(-5); };

Assert.Throws<ArgumentOutOfRangeException>(action);

}

[Fact]

public void CreateMyDictionary\_ZeroCapacity\_EmptyDictionary()

{

MyDictionary<object, object> myDictionary;

myDictionary = new MyDictionary<object, object>(0);

Assert.Empty(myDictionary);

}

[Fact]

public void CreateMyDictionary\_InitialCapacity\_ShouldBeResized()

{

int initialCapacity = 10;

var myDictionary = new MyDictionary<int, string>(initialCapacity);

for (int i = 0; i < initialCapacity + 1; i++)

{

myDictionary.Add(i, $"Value{i}");

}

Assert.True(myDictionary.Count > initialCapacity);

}

}

}

EventsTests.cs

namespace MyCollection.Tests

{

public class EventsTests

{

[Fact]

public void AddedPairEvent()

{

bool isEventTriggered = false;

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

};

myDictionary.AddedPair += (sender, args) =>

isEventTriggered = true;

myDictionary.Add(2, "Two");

Assert.True(isEventTriggered);

}

[Fact]

public void RemovedPairEvent()

{

bool isEventTriggered = false;

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

};

myDictionary.RemovedPair += (sender, args) =>

isEventTriggered = true;

myDictionary.Remove(1);

Assert.True(isEventTriggered);

}

[Fact]

public void ChangedValueEvent()

{

bool isEventTriggered = false;

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

};

myDictionary.ChangedValue += (sender, args) =>

isEventTriggered = true;

myDictionary[1] = "!ONE!";

Assert.True(isEventTriggered);

}

[Fact]

public void ClearedEvent()

{

bool isEventTriggered = false;

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" },

};

myDictionary.Cleared += (sender, args) =>

isEventTriggered = true;

myDictionary.Clear();

Assert.True(isEventTriggered);

}

}

}

IndexerTests.cs

namespace MyCollection.Tests

{

public class IndexerTest

{

[Fact]

public void GetValueIndexer\_ValueByKey1\_One()

{

var myDictionary = new Dictionary<int, string>

{

{ 1, "One" }

};

var a = myDictionary[1];

Assert.Equal("One", a);

}

[Fact]

public void GetValueIndexer\_ValueByKey2\_KeyNotFoundException()

{

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" }

};

var action = () => myDictionary[2];

Assert.Throws<KeyNotFoundException>(action);

}

[Fact]

public void SetValueIndexer\_ValueTwoByKey2\_NewItem()

{

var myDictionary = new MyDictionary<int, string>

{

{ 1, "One" }

};

myDictionary[1] = "ONE";

Assert.Equal("ONE", myDictionary[1]);

}

[Fact]

public void SetValueIndexer\_NullKey\_ArgumentNullException()

{

var myDictionary = new MyDictionary<object, object>

{

{ 1, "One" }

};

var action = () => myDictionary[null!] = "ONE";

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void SetValueIndexer\_NotExistKey2\_KeyNotFoundException()

{

var myDictionary = new MyDictionary<object, object>

{

{ 1, "One" }

};

var action = () => myDictionary[2] = "ONE";

Assert.Throws<KeyNotFoundException>(action);

}

}

}

MyEnumeratorTests.cs

namespace MyCollection.Tests

{

public class MyEnumeratorTests

{

[Fact]

public void MyEnumerator\_Current\_ValidIndex\_ReturnsKeyValuePair()

{

var myDictionary = new MyDictionary<int, string>();

myDictionary.Add(1, "One");

var enumerator = myDictionary.GetEnumerator();

enumerator.MoveNext();

var currentKeyValuePair = enumerator.Current;

Assert.Equal(new KeyValuePair<int, string>(1, "One"), currentKeyValuePair);

}

[Fact]

public void MyEnumerator\_Current\_InvalidIndex\_InvalidOperationException()

{

var myDictionary = new MyDictionary<int, string>();

var enumerator = myDictionary.GetEnumerator();

Func<object?> testCode = () => enumerator.Current;

Assert.Throws<InvalidOperationException>(testCode);

}

[Fact]

public void MyEnumerator\_Reset\_ResetsEnumeratorState()

{

// Arrange

var myDictionary = new MyDictionary<int, string>();

myDictionary.Add(1, "One");

myDictionary.Add(2, "Two");

myDictionary.Add(3, "Three");

var enumerator = myDictionary.GetEnumerator();

//Act

while (enumerator.MoveNext())

{

}

enumerator.Reset();

// Assert

Assert.True(enumerator.MoveNext());

Assert.Equal(new KeyValuePair<int, string>(1, "One"), enumerator.Current);

}

}

}

PropTests.cs

namespace MyCollection.Tests

{

public class PropTests

{

[Fact]

public void IsReadOnlyProp\_ShouldReturnFalse()

{

var myDictionary = new MyDictionary<int, string>();

bool isReadOnly = myDictionary.IsReadOnly;

Assert.False(isReadOnly);

}

[Fact]

public void CountProp\_ShouldBe3()

{

var myDictionary = new MyDictionary<int, string>()

{

{1, "one" },

{2, "two"},

{3, "three" }

};

var count = myDictionary.Count;

Assert.Equal(3, count);

}

[Fact]

public void KeysProp()

{

var myDictionary = new MyDictionary<int, string>()

{

{1, "one" },

{2, "two"},

{3, "three" }

};

var keys = myDictionary.Keys;

Assert.Collection(keys, key =>

{

Assert.Equal(1, key);

}, key =>

{

Assert.Equal(2, key);

}, key =>

{

Assert.Equal(3, key);

});

}

[Fact]

public void ValuesProp()

{

var myDictionary = new MyDictionary<int, string>()

{

{1, "one" },

{2, "two"},

{3, "three" }

};

var values = myDictionary.Values;

Assert.Collection(values, value =>

{

Assert.Equal("one", value);

}, value =>

{

Assert.Equal("two", value);

}, value =>

{

Assert.Equal("three", value);

});

}

}

}

RemovingTests.cs

namespace MyCollection.Tests

{

public class RemovingTests

{

[Fact]

public void RemoveItem\_NullKey\_ArgumentNullException()

{

var myDictionary = new MyDictionary<object, object>();

Action action = () => myDictionary.Remove(null!);

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void RemoveItem\_NotExistKey\_KeyNotFoundException()

{

var myDictionary = new MyDictionary<int, string>();

Action action = () => myDictionary.Remove(1);

Assert.Throws<KeyNotFoundException>(action);

}

[Fact]

public void RemoveItem\_ExistingKey\_RemovesKey()

{

var myDictionary = new MyDictionary<int, string>()

{

{1, "one" },

{2, "two" },

{3, "three"}

};

bool result = myDictionary.Remove(2);

Assert.True(result);

Assert.False(myDictionary.ContainsKey(2));

}

[Fact]

public void RemoveKeyValue\_NotExistPair\_ReturnsFalse()

{

var myDictionary = new MyDictionary<int, string>();

bool result = myDictionary.Remove(new KeyValuePair<int, string>(1, "One"));

Assert.False(result);

}

[Fact]

public void RemoveKeyValue\_NullKey\_ArgumentNullException()

{

var myDictionary = new MyDictionary<object, string>();

Action action = () => myDictionary.Remove(new KeyValuePair<object, string>(null!, "One"));

Assert.Throws<ArgumentNullException>(action);

}

[Fact]

public void RemoveKeyValue\_ExistingPair\_RemovesPairAndReturnsTrue()

{

var myDictionary = new MyDictionary<int, string>()

{

{ 1, "One"},

{ 2, "Two"}

};

bool result = myDictionary.Remove(new KeyValuePair<int, string>(1, "One"));

Assert.True(result);

Assert.False(myDictionary.ContainsKey(1));

}

[Fact]

public void Clear\_CountShouldBe0()

{

var myDictionary = new MyDictionary<int, string>()

{

{ 1, "One"},

{ 2, "Two"}

};

myDictionary.Clear();

Assert.Empty(myDictionary);

}

}

}

TestDataGenerator.cs

namespace MyCollection.Tests

{

internal class TestDataGenerator

{

public static IEnumerable<object[]> GetNumbersFromDataGenerator()

{

yield return new object[] {

new KeyValuePair<object, object>(1, "one" ),

new KeyValuePair<object, object>(-1, "minus one" ),

};

yield return new object[] {

new KeyValuePair<object, object>("one", 1 ),

new KeyValuePair<object, object>("ONE", 1 ),

};

yield return new object[] {

new KeyValuePair<object, object>(1.1, 1 ),

new KeyValuePair<object, object>(1.2, 1 ),

};

}

}

}

TryGetValueTests.cs

namespace MyCollection.Tests

{

public class TryGetValueTests

{

[Fact]

public void TryGetValue\_KeyExists\_ReturnsTrueAndValue()

{

var myDictionary = new MyDictionary<int, string>();

myDictionary.Add(1, "One");

bool result = myDictionary.TryGetValue(1, out var value);

Assert.True(result);

Assert.Equal("One", value);

}

[Fact]

public void TryGetValue\_KeyDoesNotExist\_ReturnsFalseAndDefault()

{

var myDictionary = new MyDictionary<int, string>();

bool result = myDictionary.TryGetValue(2, out var value);

Assert.False(result);

Assert.Null(value);

}

[Fact]

public void TryGetValue\_NullKey\_ThrowsArgumentNullException()

{

var myDictionary = new MyDictionary<object, string>();

Action action = () => myDictionary.TryGetValue(null!, out var value);

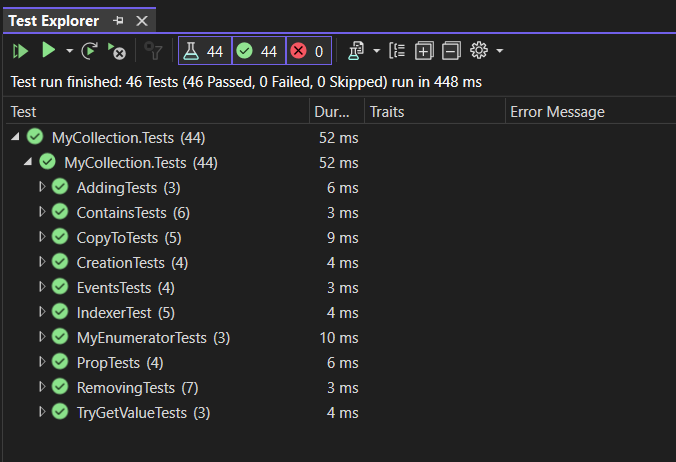
Assert.Throws<ArgumentNullException>(action);

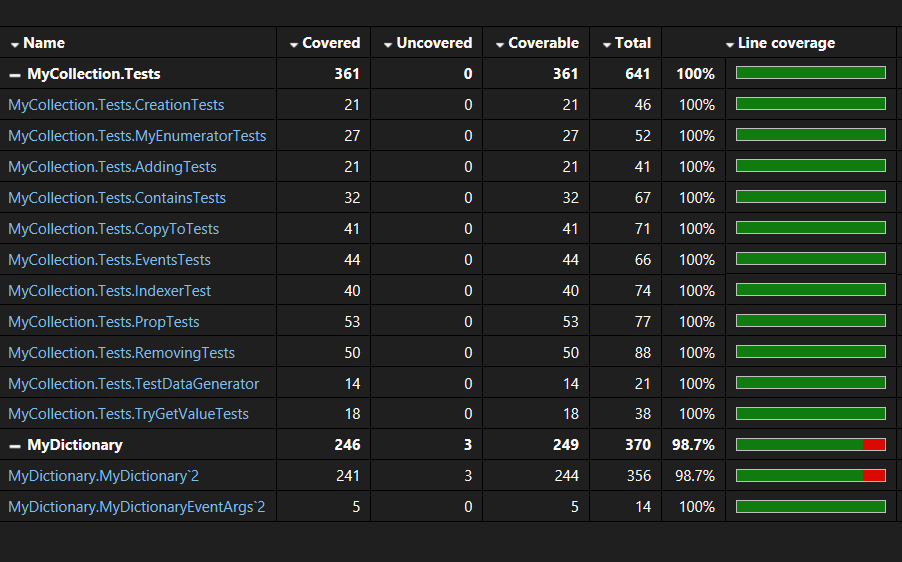
}

}

}

Результати виконання:





**Висновки:** в процесі виконання другої лабораторної роботи я навчився створювати модульні тести для вихідного коду розроблювального програмного забезпечення.