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 №3

з дисципліни

«Компоненти програмної інженерії. Якість та тестування програмного забезпечення»

на тему

«Unit тестування з використанням методів White Box Testing»

Виконав:

студент групи ІП-93

Завальнюк Максим Євгенович

Залікова книжка: 9312

Номер у списку: 10

Перевірив:

Бабарикін І. В.

Київ 2021

Мета

Написати Unit тести з використанням методів White Box Testing.

Основні кроки виконання

1. Вибрати бібліотеку для тестування за допомогою остачі від ділення номеру залікової книжки на 6.
2. Створити проект тестування, в який підключити бібліотеку.
3. Написати юніт-тести притримуючись методів White Box Testing.
4. Підготувати звіт про виконану роботу, який буде містити використані методи White Box Testing, сирцеві коди юніт-тестів та/або посилання на GitHub де розміщено проект, результати тестування і їх аналіз.

Виконання роботи

Для початку я обрахував свій варіант – 9312 mod 6 = 0. Отже, бібліотека, яку необхідно мені тестувати – BinaryFlag.

Буду я працювати із платформою **.NET 5** та бібліотекою для тестування **xUnit**.

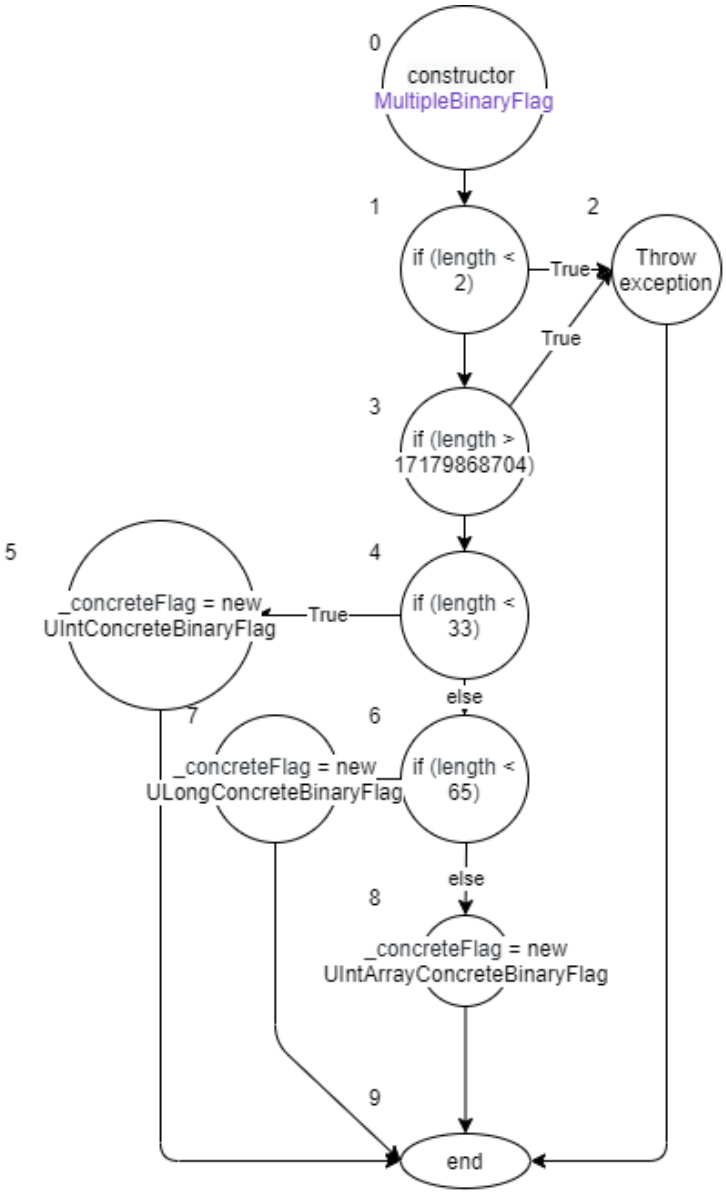
Далі я створив проект Lab3. До проекту я під'єднав файл **.dll** бібліотеки, що тестую.

Оскільки це WhiteBox тестування, то я у мене була змога подивитись код бібліотеки. У ньому я побачив один публічний клас з декількома методами – MultipleBinaryFlag. Тому тестувати буду його.

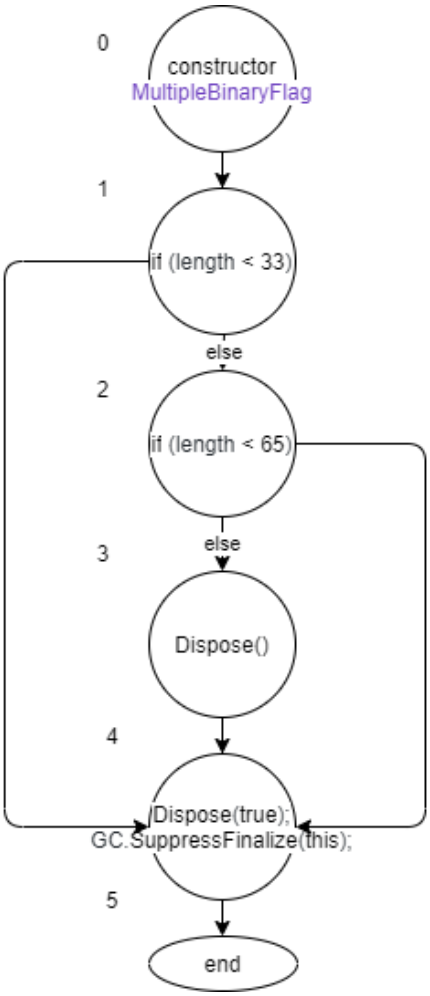
Далі необхідно обрати техніку **WBT** і відповідно до неї написати тести. Я обрав **Тестування потоку виконання**. Коли програма тестується даним видом, то відповідні тест-кейси створюються таким чином, щоб перевірити правильність виконання максимально можливої кількість шляхів виконання ПЗ. Отже, моїм завданням стало створення таких шляхів, і найкраще це можна представити графічно.

Я створив діаграми для кожного методу, що тестую, щоб побачити усі можливі «сценарії»:

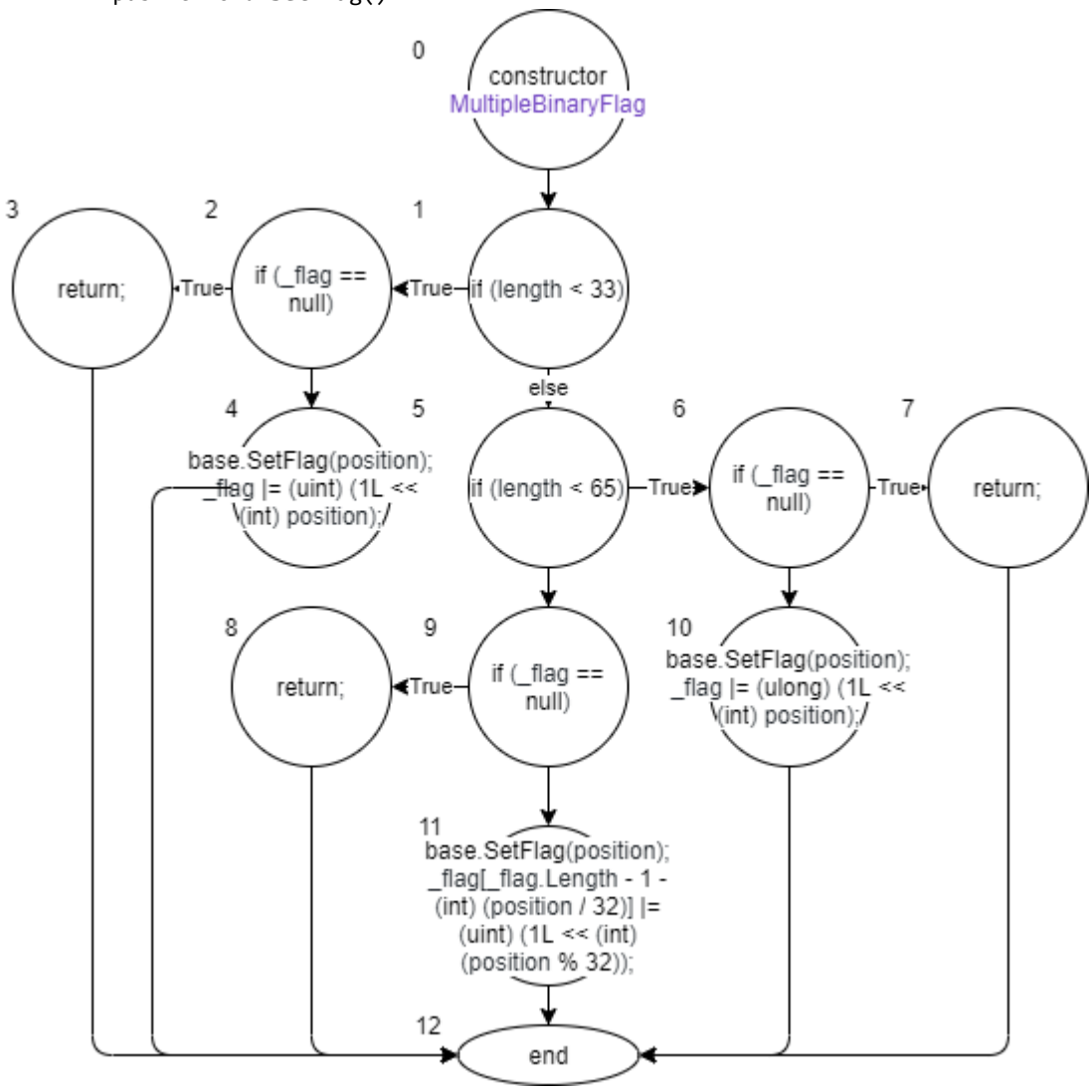
- `public MultipleBinaryFlag()`



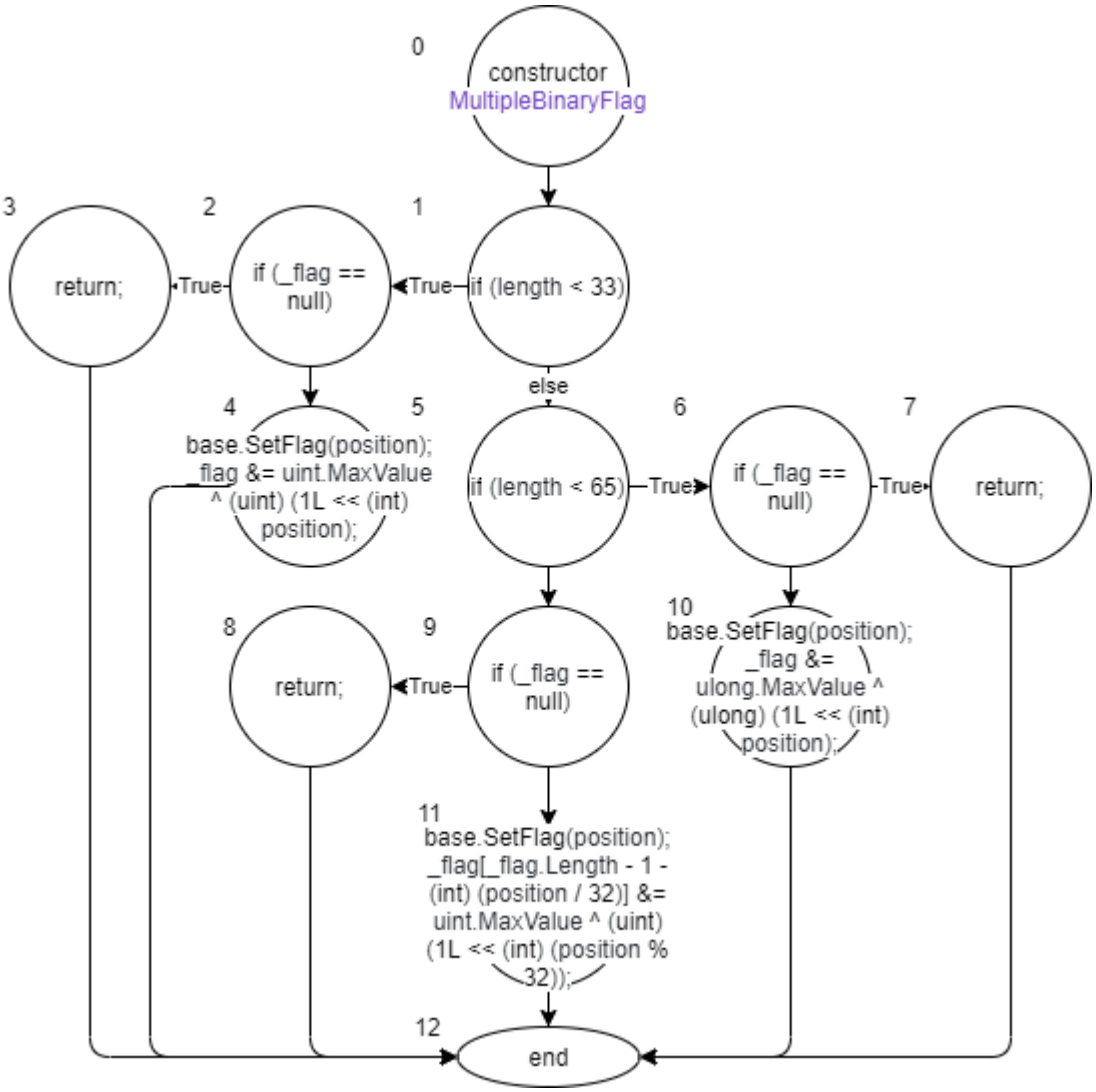
- `public void Dispose()`



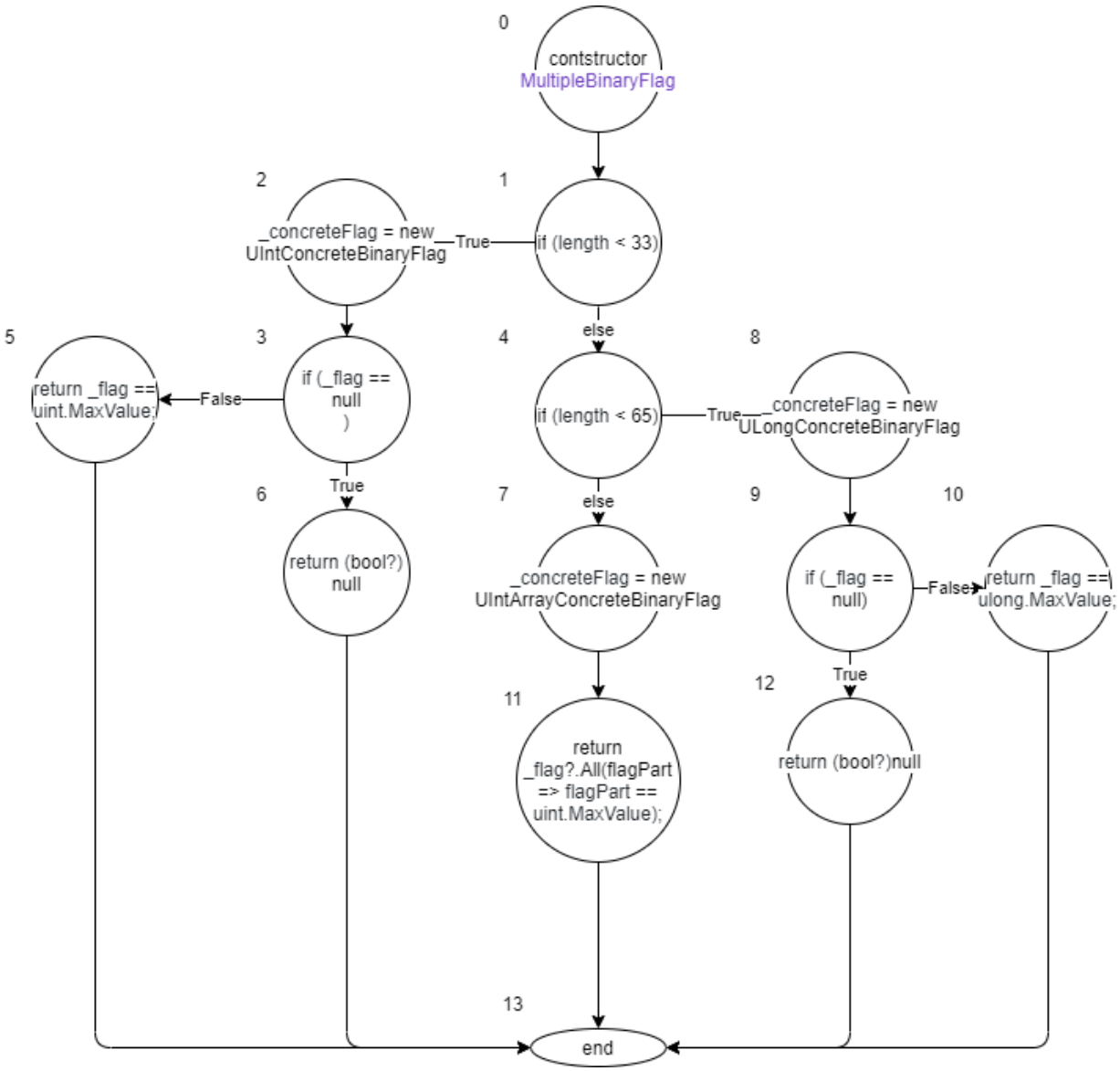
- `public void SetFlag()`



- `public void ResetFlag()`



- `public bool? GetFlag()`



Усі ці діаграми знаходяться [тут](#).

Тепер по заданим маршрутам ми створюємо тест-кейси:

```
using IIG.BinaryFlag;
using System;
using Xunit;

namespace Lab3
{
    public class MultipleBinaryFlagTests
    {
        /// <summary>
        /// Test for constructor method
        /// </summary>
        public class ConstructorTests
        {
            /// <summary>
```

```
/// Test UIntArrayConcreteBinaryFlag object creation
/// </summary>
[Fact]
public void Route_0_1_3_4_6_8_9true()
{
    try
    {
        MultipleBinaryFlag obj = new(65 + 1, true);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test UIntArrayConcreteBinaryFlag object creation
/// </summary>
[Fact]
public void Route_0_1_3_4_6_8_9false()
{
    try
    {
        MultipleBinaryFlag obj = new(65 + 1, false);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ULongConcreteBinaryFlag object creation
/// </summary>
[Fact]
public void Route_0_1_3_4_6_7_9true()
{
    try
    {
        MultipleBinaryFlag obj = new(64, true);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ULongConcreteBinaryFlag object creation
/// </summary>
```

```
[Fact]
public void Route_0_1_3_4_6_7_9false()
{
    try
    {
        MultipleBinaryFlag obj = new(64, false);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test UIntConcreteBinaryFlag object creation
/// </summary>
[Fact]
public void Route_0_1_3_4_5_9true()
{
    try
    {
        MultipleBinaryFlag obj = new(32, true);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test UIntConcreteBinaryFlag object creation
/// </summary>
[Fact]
public void Route_0_1_3_4_5_9false()
{
    try
    {
        MultipleBinaryFlag obj = new(32, false);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ArgumentOutOfRangeException exception
/// </summary>
[Fact]
public void Route_0_1_3_2_9true()
```

```
{
    try
    {
        MultipleBinaryFlag obj = new(17179868704 + 1, true);
        Assert.False(true);
    }
    catch (ArgumentOutOfRangeException)
    {
        Assert.True(true);
    }
}
/// <summary>
/// Test ArgumentOutOfRangeException exception
/// </summary>
[Fact]
public void Route_0_1_3_2_9false()
{
    try
    {
        MultipleBinaryFlag obj = new(17179868704 + 1, false);
        Assert.False(true);
    }
    catch (ArgumentOutOfRangeException)
    {
        Assert.True(true);
    }
}
/// <summary>
/// Test ArgumentOutOfRangeException exception
/// </summary>
[Fact]
public void Route_0_1_2_9true()
{
    try
    {
        MultipleBinaryFlag obj = new(2 - 1, true);
        Assert.False(true);
    }
    catch (ArgumentOutOfRangeException)
    {
        Assert.True(true);
    }
}
/// <summary>
/// Test ArgumentOutOfRangeException exception
/// </summary>
[Fact]
public void Route_0_1_2_9false()
{
    try
```

```
        {
            MultipleBinaryFlag obj = new(2 - 1, false);
            Assert.False(true);
        }
        catch (ArgumentOutOfRangeException)
        {
            Assert.True(true);
        }
    }
}

public class GetFlagMethodTests
{
    /// <summary>
    /// Test GetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_4_7_11_13true()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 1, true);
            bool? expected = obj.GetFlag();
            Assert.NotNull(expected);
            Assert.True(expected);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test GetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_4_7_11_13false()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 1, false);
            bool? expected = obj.GetFlag();
            Assert.NotNull(expected);
            Assert.False(expected);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
```

```
/// Test GetFlag method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_4_8_9_10_13true()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, true);
        bool? expected = obj.GetFlag();
        Assert.NotNull(expected);
        Assert.True(expected);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test GetFlag method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_4_8_9_10_13false()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, false);
        bool? expected = obj.GetFlag();
        Assert.NotNull(expected);
        Assert.False(expected);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test GetFlag method with UIntConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_2_3_5_13true()
{
    try
    {
        MultipleBinaryFlag obj = new(33 - 1, true);
        bool? expected = obj.GetFlag();
        Assert.NotNull(expected);
        Assert.True(expected);
    }
    catch (Exception)
    {
```

```
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test GetFlag method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_3_5_13false()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, false);
            bool? expected = obj.GetFlag();
            Assert.NotNull(expected);
            Assert.False(expected);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
}
public class SetFlagMethodTests
{
    /// <summary>
    /// Test SetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_5_9_11_12true()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 123, true);
            obj.SetFlag(100);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test SetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_5_9_11_12false()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 123, false);
```

```
            obj.SetFlag(100);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test SetFlag method with ULongConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_5_6_10_12true()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 - 1, true);
            obj.SetFlag(50);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test SetFlag method with ULongConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_5_6_10_12false()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 - 1, false);
            obj.SetFlag(50);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test SetFlag method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_4_12true()
    {
        try
        {
```

```
            MultipleBinaryFlag obj = new(33 - 1, true);
            obj.SetFlag(20);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test SetFlag method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_4_12false()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, false);
            obj.SetFlag(20);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
}
public class ResetFlagMethodTests
{
    /// <summary>
    /// Test ResetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_5_9_11_12true()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 123, true);
            obj.ResetFlag(100);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test ResetFlag method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
```

```
public void Route_0_1_5_9_11_12false()
{
    try
    {
        MultipleBinaryFlag obj = new(65 + 123, false);
        obj.ResetFlag(100);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ResetFlag method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_5_6_10_12true()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, true);
        obj.ResetFlag(50);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ResetFlag method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_5_6_10_12false()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, false);
        obj.ResetFlag(50);
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ResetFlag method with UIntConcreteBinaryFlag object
/// </summary>
```

```
    [Fact]
    public void Route_0_1_2_4_12true()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, true);
            obj.ResetFlag(20);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test ResetFlag method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_4_12false()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, false);
            obj.ResetFlag(20);
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
}
public class DisposeMethodTests
{
    /// <summary>
    /// Test Dispose method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_4_5true()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, true);
            obj.Dispose();
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
```

```
}
/// <summary>
/// Test Dispose method with UIntConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_4_5false()
{
    try
    {
        MultipleBinaryFlag obj = new(33 - 1, false);
        obj.Dispose();
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test Dispose method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_2_4_5true()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, true);
        obj.Dispose();
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test Dispose method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void Route_0_1_2_4_5false()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, false);
        obj.Dispose();
        Assert.True(true);
    }
    catch (Exception)
    {
        Assert.False(true);
```

```
        }
    }
    /// <summary>
    /// Test Dispose method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_3_4_5true()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 1, true);
            obj.Dispose();
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
    /// <summary>
    /// Test Dispose method with UIntArrayConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void Route_0_1_2_3_4_5false()
    {
        try
        {
            MultipleBinaryFlag obj = new(65 + 1, false);
            obj.Dispose();
            Assert.True(true);
        }
        catch (Exception)
        {
            Assert.False(true);
        }
    }
}
public class ToStringMethodTests
{
    /// <summary>
    /// Test ToString method with UIntConcreteBinaryFlag object
    /// </summary>
    [Fact]
    public void ToString_UIntConcreteBinaryFlag_NotNulltrue()
    {
        try
        {
            MultipleBinaryFlag obj = new(33 - 1, true);
            string expected = obj.ToString();
            Assert.NotNull(expected);
```

```
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ToString method with UIntConcreteBinaryFlag object
/// </summary>
[Fact]
public void ToString_UIntConcreteBinaryFlag_NotNullfalse()
{
    try
    {
        MultipleBinaryFlag obj = new(33 - 1, false);
        string expected = obj.ToString();
        Assert.NotNull(expected);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ToString method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void ToString_ULongConcreteBinaryFlag_NotNulltrue()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, true);
        string expected = obj.ToString();
        Assert.NotNull(expected);
    }
    catch (Exception)
    {
        Assert.False(true);
    }
}
/// <summary>
/// Test ToString method with ULongConcreteBinaryFlag object
/// </summary>
[Fact]
public void ToString_ULongConcreteBinaryFlag_NotNullfalse()
{
    try
    {
        MultipleBinaryFlag obj = new(65 - 1, false);
        string expected = obj.ToString();
```

```
                Assert.NotNull(expected);
            }
            catch (Exception)
            {
                Assert.False(true);
            }
        }
        /// <summary>
        /// Test ToString method with UIntArrayConcreteBinaryFlag object
        /// </summary>
        [Fact]
        public void ToString_UIntArrayConcreteBinaryFlag_NotNulltrue()
        {
            try
            {
                MultipleBinaryFlag obj = new(65 + 1, true);
                string expected = obj.ToString();
                Assert.NotNull(expected);
            }
            catch (Exception)
            {
                Assert.False(true);
            }
        }
        /// <summary>
        /// Test ToString method with UIntArrayConcreteBinaryFlag object
        /// </summary>
        [Fact]
        public void ToString_UIntArrayConcreteBinaryFlag_NotNullfalse()
        {
            try
            {
                MultipleBinaryFlag obj = new(65 + 1, false);
                string expected = obj.ToString();
                Assert.NotNull(expected);
            }
            catch (Exception)
            {
                Assert.False(true);
            }
        }
    }
}
```

У назвах методів вказано, який сценарій я тестую, і перевірити його можна по діаграмі. Тільки при тестуванні методу **ToString** немає сценаріїв, оскільки вони там недоречні.

Висновок

Під час виконання роботи я познайомився із технологією **WhiteBox** тестування. Вона істотно відрізняється від **BBT.** Головним завданням є правильне складання усіх тест-кейсів, тому це не легше, ніж саме написання тестів.

Джерела

1. Папка з програмою.
2. Папка з діаграмами.
3. Результати CI.