**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: **«**Паттерны проектирования. Паттерн «Декоратор»**»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2024

**Цель работы**: изучить основы шаблонного проектирования, получить навыки работы с паттернами (на примере паттерна «Декоратор»). Продемонстрировать работу декорируемого класса (класса *Stream*) на 3-х классах семейства *Stream* (*FileStream*, *MemoryStream*, *BufferStream*).

**Задание:**

• Разработать библиотеку классов для декорирования класса *Stream*согласно варианту**.**

• Разработать графическое приложение *WinForm*/*WPF* (согласовать с преподавателем) для демонстрации работы декорируемого класса. Продемонстрировать работу декораций минимум на 3-х классах семейства *Stream* (*FileStream*, *MemoryStream*, *BufferStream* и др.).

• Приложение должно быть простым в использовании и включать в себя полную обработку исключений.

• При написании и оформлении кода обязательно руководствоваться *Code* *Convention*, принципами ООП, *SOLID* и использовать элементы авто документирования с генерацией соответствующих файлов.

• Написать *unit*-тесты и исходные файлы для тестирования разработанных библиотечных классов, тестирование должно покрывать более 80% библиотечного кода.

|  |  |
| --- | --- |
| № | Условие |
| 14 | Подсчитывает общее количество записанных байт (*Write*) |

**Ход работы**

На рисунке 1 представлено главное окно приложения.

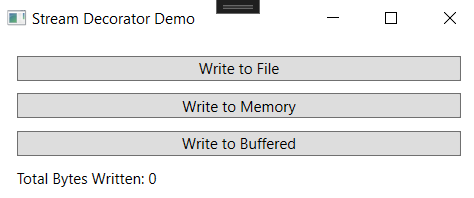


Рисунок 1 – Главное меню

При нажатии на первую кнопку в текстовый файл «*testfile.txt*». Результат представлен на рисунке 2.

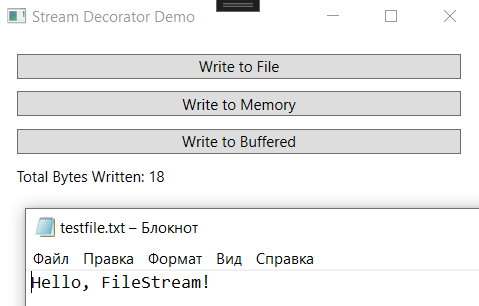


Рисунок 2 – Результат работы при выборе первой опции

При выборе третьей опции произойдёт то же самое: в файл будет записана строка, но уже «*Hello, BufferedStream*!» Однако при выборе функции «*Write to Memory*» данные не будут записаны в текстовый документ, но будут отображены в виде диалогового окна. Это показано на рисунке 3.

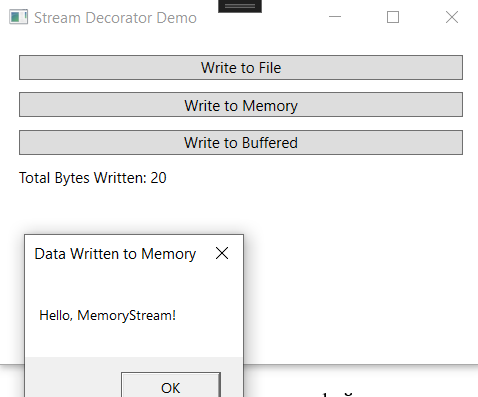


Рисунок 3 – Диалоговое окно при выборе второй функции

На рисунке 4 представлены результаты *unit*-тестов.

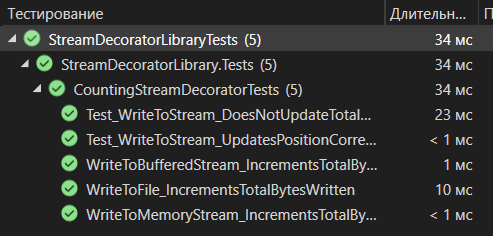


Рисунок 4 – Выполненные модульные тесты

В приложении А представлен код программы.

На рисунке 5 представлен репозиторий проекта.

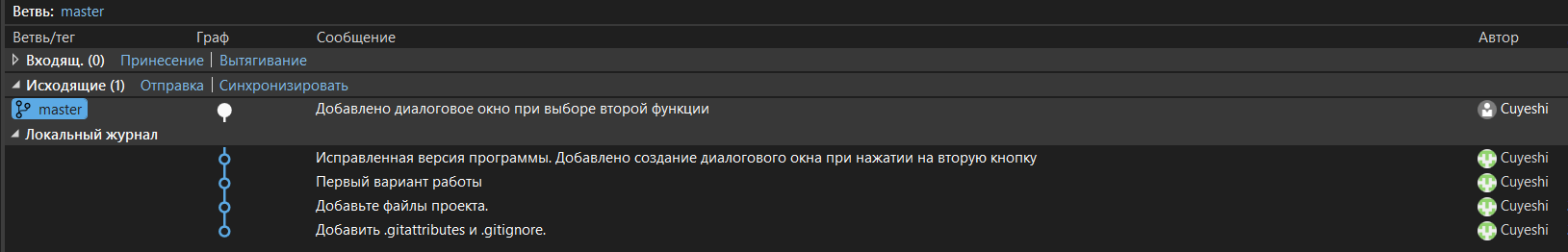


Рисунок 5 – Репозиторий проекта

**Вывод:** в результате выполнения лабораторной работы получили новые (и закрепили уже имеющиеся) знания работы со средой разработки *Visual Studio*, изучили основышаблонного проектирования, получили навыки работы с паттерном «Декоратор» и продемонстрировали его работу на примере 3-х классов семейства *Stream* (*FileStream*, *MemoryStream*, *BufferStream*).

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**MainWindow.xaml**

<Window x:Class="StreamDecoratorDemo.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Stream Decorator Demo" Width="400" Height="300">

<Grid>

<StackPanel Margin="10">

<Button Content="Write to File" Click="WriteToFile\_Click" Margin="5"/>

<Button Content="Write to Memory" Click="WriteToMemory\_Click" Margin="5"/>

<Button Content="Write to Buffered" Click="WriteToBuffered\_Click" Margin="5"/>

<TextBlock x:Name="TotalBytesWrittenText" Text="Total Bytes Written: 0" Margin="5"/>

</StackPanel>

</Grid>

</Window>

**MainWindow.xaml.cs**

using System.IO;

using System.Text;

using System.Windows;

using StreamDecoratorLibrary;

namespace StreamDecoratorDemo

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void UpdateTotalBytesWritten(long totalBytesWritten)

{

TotalBytesWrittenText.Text = $"Total Bytes Written: {totalBytesWritten}";

}

private void WriteToFile\_Click(object sender, RoutedEventArgs e)

{

string filePath = "testfile.txt";

using (FileStream fileStream = new FileStream(filePath, FileMode.Create))

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(fileStream))

{

byte[] data = Encoding.UTF8.GetBytes("Hello, FileStream!");

countingStream.Write(data, 0, data.Length);

UpdateTotalBytesWritten(countingStream.TotalBytesWritten);

}

}

private void WriteToMemory\_Click(object sender, RoutedEventArgs e)

{

using (MemoryStream memoryStream = new MemoryStream())

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(memoryStream))

{

byte[] data = Encoding.UTF8.GetBytes("Hello, MemoryStream!");

countingStream.Write(data, 0, data.Length);

UpdateTotalBytesWritten(countingStream.TotalBytesWritten);

// Прочитать данные из MemoryStream и отобразить их в текстовом блоке

memoryStream.Seek(0, SeekOrigin.Begin);

StreamReader reader = new StreamReader(memoryStream);

string content = reader.ReadToEnd();

MessageBox.Show(content, "Data Written to Memory");

}

}

private void WriteToBuffered\_Click(object sender, RoutedEventArgs e)

{

string filePath = "testfile.txt";

using (FileStream fileStream = new FileStream(filePath, FileMode.Create))

using (BufferedStream bufferedStream = new BufferedStream(fileStream))

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(bufferedStream))

{

byte[] data = Encoding.UTF8.GetBytes("Hello, BufferedStream!");

countingStream.Write(data, 0, data.Length);

UpdateTotalBytesWritten(countingStream.TotalBytesWritten);

}

}

}

}

**CountingStreamDecorator.cs**

using System.IO;

namespace StreamDecoratorLibrary

{

public class CountingStreamDecorator : Stream

{

private readonly Stream \_stream;

private long \_totalBytesWritten;

public CountingStreamDecorator(Stream stream)

{

\_stream = stream;

\_totalBytesWritten = 0;

}

public long TotalBytesWritten => \_totalBytesWritten;

public override bool CanRead => \_stream.CanRead;

public override bool CanSeek => \_stream.CanSeek;

public override bool CanWrite => \_stream.CanWrite;

public override long Length => \_stream.Length;

public override long Position

{

get => \_stream.Position;

set => \_stream.Position = value;

}

public override void Flush() => \_stream.Flush();

public override int Read(byte[] buffer, int offset, int count) => \_stream.Read(buffer, offset, count);

public override long Seek(long offset, SeekOrigin origin) => \_stream.Seek(offset, origin);

public override void SetLength(long value) => \_stream.SetLength(value);

public override void Write(byte[] buffer, int offset, int count)

{

\_stream.Write(buffer, offset, count);

\_totalBytesWritten += count;

}

protected override void Dispose(bool disposing)

{

if (disposing)

\_stream.Dispose();

base.Dispose(disposing);

}

}

}

**CountingStreamDecoratorTests.cs**

using System;

using System.IO;

using System.Text;

using StreamDecoratorLibrary;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace StreamDecoratorLibrary.Tests

{

[TestClass]

public class CountingStreamDecoratorTests

{

[TestMethod]

public void WriteToFile\_IncrementsTotalBytesWritten()

{

// Arrange

string filePath = "testfile.txt";

using (FileStream fileStream = new FileStream(filePath, FileMode.Create))

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(fileStream))

{

byte[] data = Encoding.UTF8.GetBytes("Test data");

long initialBytesWritten = countingStream.TotalBytesWritten;

// Act

countingStream.Write(data, 0, data.Length);

// Assert

Assert.AreEqual(initialBytesWritten + data.Length, countingStream.TotalBytesWritten);

}

}

[TestMethod]

public void WriteToMemoryStream\_IncrementsTotalBytesWritten()

{

// Arrange

using (MemoryStream memoryStream = new MemoryStream())

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(memoryStream))

{

byte[] data = Encoding.UTF8.GetBytes("Test data");

long initialBytesWritten = countingStream.TotalBytesWritten;

// Act

countingStream.Write(data, 0, data.Length);

// Assert

Assert.AreEqual(initialBytesWritten + data.Length, countingStream.TotalBytesWritten);

}

}

[TestMethod]

public void WriteToBufferedStream\_IncrementsTotalBytesWritten()

{

// Arrange

string filePath = "testfile.txt";

using (FileStream fileStream = new FileStream(filePath, FileMode.Create))

using (BufferedStream bufferedStream = new BufferedStream(fileStream))

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(bufferedStream))

{

byte[] data = Encoding.UTF8.GetBytes("Test data");

long initialBytesWritten = countingStream.TotalBytesWritten;

// Act

countingStream.Write(data, 0, data.Length);

// Assert

Assert.AreEqual(initialBytesWritten + data.Length, countingStream.TotalBytesWritten);

}

}

[TestMethod]

public void Test\_WriteToStream\_UpdatesPositionCorrectly()

{

// Arrange

using (MemoryStream memoryStream = new MemoryStream())

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(memoryStream))

{

byte[] data1 = Encoding.UTF8.GetBytes("Test data");

byte[] data2 = Encoding.UTF8.GetBytes("Additional test data");

// Act

countingStream.Write(data1, 0, data1.Length);

long positionBefore = countingStream.Position;

countingStream.Write(data2, 0, data2.Length);

long positionAfter = countingStream.Position;

// Assert

Assert.AreEqual(data1.Length, positionBefore);

Assert.AreEqual(data1.Length + data2.Length, positionAfter);

}

}

[TestMethod]

public void Test\_WriteToStream\_DoesNotUpdateTotalBytesWrittenUntilFlush()

{

// Arrange

using (MemoryStream memoryStream = new MemoryStream())

using (CountingStreamDecorator countingStream = new CountingStreamDecorator(memoryStream))

{

byte[] data = Encoding.UTF8.GetBytes("Test data");

// Act

countingStream.Write(data, 0, data.Length);

// Assert

Assert.AreEqual(9, countingStream.TotalBytesWritten);

// Flush the stream

countingStream.Flush();

// Assert

Assert.AreEqual(data.Length, countingStream.TotalBytesWritten);

}

}

}

}