**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

Бахонько А. М.

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 |
| 2 | Подсчитывает время, сколько поток не использовался. |

**Ход работы**

На рисунке 1 представлено главное окно приложения.

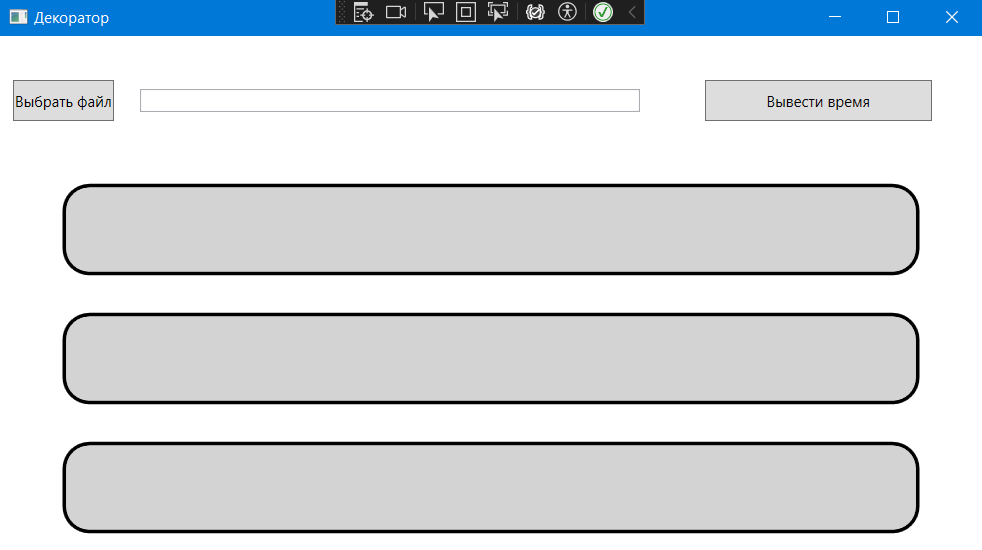


Рисунок 1 – Главное меню

Выберем файл, с которого хотим считать данные. Результат представлен на рисунке 2.



Рисунок 2 – Выбор файла

Содержимое выбранного файла представлено на рисунке 3.



Рисунок 3 – Содержимое выбранного файла

Теперь нажмем кнопку «Вывести время» и увидим время неиспользования потока для каждого класса семейства *Stream*. Результат представлен на рисунке 4.

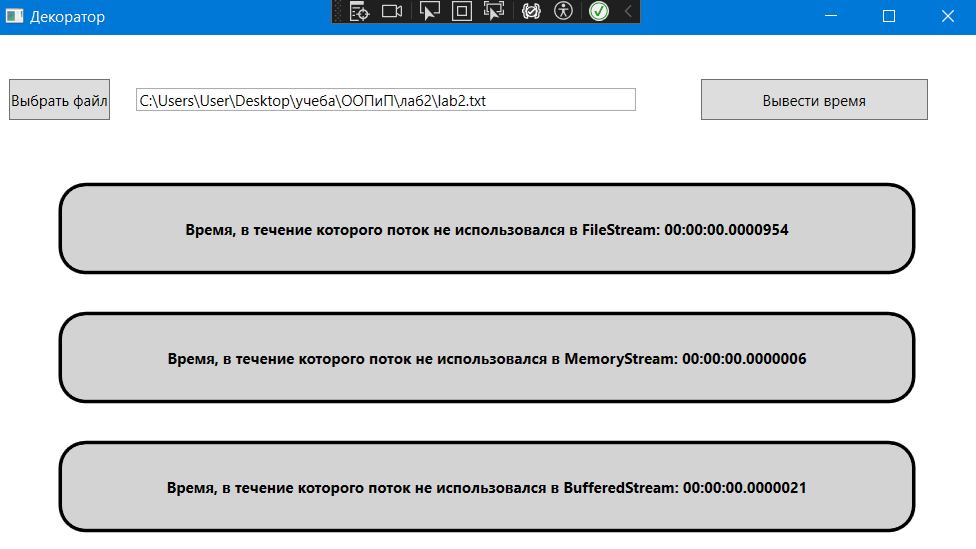


Рисунок 4 – Результат работы программы

Также предусмотрены ситуации, в которых пользователь не выберет файл. Результат представлен на рисунках 5.

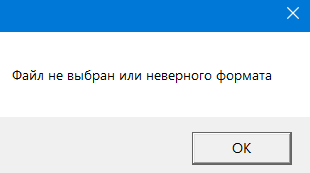


Рисунок 5 – Информационное окно, появляющееся в случае отсутствия пути к файлу

На рисунке 6 представлены результаты *unit*-тестов.

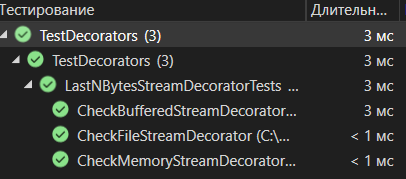


Рисунок 6 – Выполненные модульные тесты

В приложении А представлен код программы.

На рисунке 7 представлен репозиторий проекта.

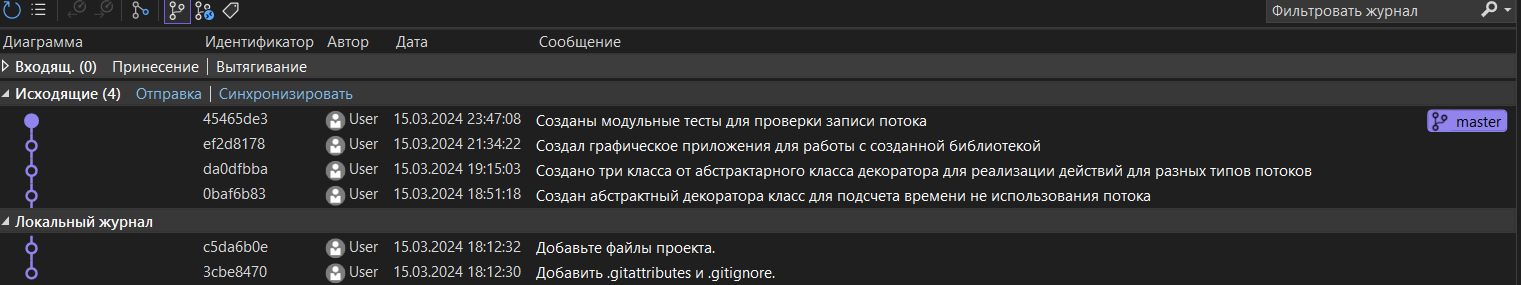


Рисунок 7 – Репозиторий проекта

**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="laba2.MainWindow"

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

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

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:laba2"

mc:Ignorable="d"

Title="Декоратор" Height="450" Width="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid Grid.Row="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="50\*"/>

<ColumnDefinition Width="25\*"/>

</Grid.ColumnDefinitions>

<Button Margin="10,35,10,35" HorizontalAlignment="Left" Grid.Column="0" Click="Button\_Click">Выбрать файл</Button>

<TextBox x:Name="TextBoxFilePath" VerticalAlignment="Center" Margin="100,0,0,0" IsReadOnly="True" Width="400"/>

<Button HorizontalContentAlignment="Center" Margin="40,35,40,35" Grid.Column="2" Content="Вывести время" Click="GetTime"/>

</Grid>

<Grid Grid.Row="1">

<Border CornerRadius="20" Background="LightGray" Margin="50,15,50,15" BorderBrush="Black" BorderThickness="3">

<TextBlock x:Name="FirstResultTextBlock" VerticalAlignment="Center" HorizontalAlignment="Center" FontWeight="Bold"/>

</Border>

</Grid>

<Grid Grid.Row="2">

<Border CornerRadius="20" Background="LightGray" Margin="50,15,50,15" BorderBrush="Black" BorderThickness="3">

<TextBlock x:Name="SecondResultTextBlock" VerticalAlignment="Center" HorizontalAlignment="Center" FontWeight="Bold"/>

</Border>

</Grid>

<Grid Grid.Row="3">

<Border CornerRadius="20" Background="LightGray" Margin="50,15,50,15" BorderBrush="Black" BorderThickness="3">

<TextBlock x:Name="ThirdResultTextBlock" VerticalAlignment="Center" HorizontalAlignment="Center" FontWeight="Bold"/>

</Border>

</Grid>

</Grid>

</Window>

**MainWindow.xaml.cs**

using LibraryDecorator;

using Microsoft.Win32;

using System.IO;

using System.Text.RegularExpressions;

using System.Windows;

namespace laba2

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private int n = 10;

private string filePath = "C:\\Users\\User\\Desktop\\учеба\\ООПиП\\лаб2\\lab2.txt";

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

{

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true)

{

TextBoxFilePath.Text = openFileDialog.FileName;

}

}

private void GetTime(object sender, RoutedEventArgs e)

{

if (!string.IsNullOrEmpty(TextBoxFilePath.Text) && Regex.IsMatch(TextBoxFilePath.Text, "\\.txt$"))

{

filePath = TextBoxFilePath.Text;

}

else

{

MessageBox.Show("Файл не выбран или неверного формата");

return;

}

try

{

using (var fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

byte[] bufferFull = new byte[fileStream.Length];

var decorator = new FileStreamDecorator(fileStream);

int bytesRead = decorator.Read(bufferFull, 0, (int)fileStream.Length);

if (bytesRead == 0)

{

MessageBox.Show("Файл пустой");

return;

}

FirstResultTextBlock.Text = "Время, в течение которого поток не использовался в FileStream: " + decorator.GetTime();

}

using (var memoryStream = new MemoryStream())

{

using (var fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

fileStream.CopyTo(memoryStream);

}

byte[] bufferFull = new byte[memoryStream.Length];

memoryStream.Seek(0, SeekOrigin.Begin);

var decorator = new MemoryStreamDecorator(memoryStream);

int bytesRead = decorator.Read(bufferFull, 0, (int)memoryStream.Length);

if (bytesRead == 0)

{

MessageBox.Show("Файл пустой");

return;

}

SecondResultTextBlock.Text = "Время, в течение которого поток не использовался в MemoryStream: " + decorator.GetTime();

}

using (var fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

using (var bufferStream = new BufferedStream(fileStream))

{

byte[] bufferFull = new byte[bufferStream.Length];

var decorator = new BufferedStreamDecorator(bufferStream);

int bytesRead = decorator.Read(bufferFull, 0, (int)bufferStream.Length);

if (bytesRead == 0)

{

MessageBox.Show("Файл пустой");

return;

}

ThirdResultTextBlock.Text = "Время, в течение которого поток не использовался в BufferedStream: " + decorator.GetTime();

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка: {ex.Message}");

}

}

}

}

**StreamDecorator.cs**

namespace LibraryDecorator

{

public abstract class StreamDecorator : Stream

{

/// <summary>

/// Поле для хранения объекта класса Stream

/// </summary>

private Stream \_stream;

/// <summary>

/// Поле для хранения времени

/// </summary>

private TimeSpan \_time;

private DateTime \_start = DateTime.Now;

public override bool CanRead => throw new NotImplementedException();

public override bool CanSeek => throw new NotImplementedException();

public override bool CanWrite => throw new NotImplementedException();

public override long Length => throw new NotImplementedException();

public override long Position { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

/// <summary>

/// Конструктор, принимающий объект класса Stream

/// </summary>

/// <param name="stream">Поток передаваемый в конструктор</param>

/// <exception cref="ArgumentNullException"></exception>

public StreamDecorator(Stream stream)

{

\_stream = stream ?? throw new ArgumentNullException(nameof(stream));

}

/// <summary>

/// Метод для получения времени

/// </summary>

/// <returns></returns>

public TimeSpan GetTime()

{

return \_time;

}

public override void Flush()

{

throw new NotImplementedException();

}

/// <summary>

/// Метод чтения потока с подсчётом времени неиспользования

/// </summary>

/// <param name="buffer"></param>

/// <param name="offset"></param>

/// <param name="count"></param>

/// <returns></returns>

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

{

// Записываем текущее время перед вызовом метода Read у объекта класса Stream

DateTime end = DateTime.Now;

// Вызываем метод Read у объекта класса Stream

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

// Вычисляем время, потраченное на чтение из потока

\_time = end - \_start;

\_start = end;

return count;

}

public override long Seek(long offset, SeekOrigin origin)

{

throw new NotImplementedException();

}

public override void SetLength(long value)

{

throw new NotImplementedException();

}

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

{

// Записываем текущее время перед вызовом метода Write у объекта класса Stream

DateTime end = DateTime.Now;

// Вызываем метод Write у объекта класса Stream

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

// Вычисляем время, потраченное на запись в поток

\_time = end - \_start;

\_start = end;

}

}

}

**FileStreamDecorator.cs**

namespace LibraryDecorator

{

public class FileStreamDecorator : StreamDecorator

{

public FileStreamDecorator(Stream stream) : base(stream) { }

}

}

**MemoryStreamDecorator.cs**

namespace LibraryDecorator

{

public class MemoryStreamDecorator : StreamDecorator

{

public MemoryStreamDecorator(Stream stream) : base(stream) { }

}

}

**BufferedStreamDecorator.cs**

namespace LibraryDecorator

{

public class BufferedStreamDecorator : StreamDecorator

{

public BufferedStreamDecorator(Stream stream) : base(stream) { }

}

}

**TestStream.cs**

using LibraryDecorator;

namespace TestDecorators

{

[TestClass]

public class TestsStream

{

[TestMethod]

[DataRow("C:\\Users\\User\\Desktop\\учеба\\ООПиП\\лаб2\\lab2.txt")]

public void CheckFileStreamDecorator(string filePath)

{

using (var \_fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

var \_decoratorFileStream = new FileStreamDecorator(\_fileStream);

byte[] buffer = { 118, 119, 120, 121, 122 };

var start = \_decoratorFileStream.GetTime();

\_decoratorFileStream.Write(buffer, 0, buffer.Length);

var end = \_decoratorFileStream.GetTime();

Assert.AreNotEqual(start, end);

}

}

[TestMethod]

[DataRow("C:\\Users\\User\\Desktop\\учеба\\ООПиП\\лаб2\\lab2.txt")]

public void CheckMemoryStreamDecorator(string filePath)

{

using (var memoryStream = new MemoryStream())

{

using (var fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

fileStream.CopyTo(memoryStream);

}

memoryStream.Seek(0, SeekOrigin.Begin);

var \_decoratorMemoryStream = new MemoryStreamDecorator(memoryStream);

byte[] buffer = { 118, 119, 120, 121, 122 };

var start = \_decoratorMemoryStream.GetTime();

\_decoratorMemoryStream.Write(buffer, 0, buffer.Length);

var end = \_decoratorMemoryStream.GetTime();

Assert.AreNotEqual(start, end);

}

}

[TestMethod]

[DataRow("C:\\Users\\User\\Desktop\\учеба\\ООПиП\\лаб2\\lab2.txt")]

public void CheckBufferedStreamDecorator(string filePath)

{

using (var fileStream = new FileStream(filePath, FileMode.OpenOrCreate))

{

using (var bufferStream = new BufferedStream(fileStream))

{

var \_decoratorBufferedStream = new BufferedStreamDecorator(bufferStream);

byte[] buffer = { 118, 119, 120, 121, 122 };

var start = \_decoratorBufferedStream.GetTime();

\_decoratorBufferedStream.Write(buffer, 0, buffer.Length);

var end = \_decoratorBufferedStream.GetTime();

Assert.AreNotEqual(start, end);

}

}

}

}

}