Program.cs

using System.Security.Cryptography; // Подключаем пространство имен для работы с криптографией

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Xml; // Подключаем пространство имен для работы с XML

using System.Xml.Linq; // Подключаем пространство имен для работы с LINQ to XML

using Lec5Lib; // Подключаем пользовательскую библиотеку с интерфейсами и классами

namespace lab5 // Определяем пространство имен lab5

{

internal class Program // Определяем внутренний класс Program

{

static void Main(string[] args) // Определяем метод Main, точку входа в программу

{

Console.WriteLine("Лабораторная работа # 5"); // Выводим название лабораторной работы на консоль

{ // Начало первого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecSHA512(rsa); // Создаем экземпляр DecSHA512 для хэширования с использованием rsa

string? result = hash.Save("AAAAABBBBCCCCC"); // Хэшируем строку и сохраняем результат

bool testresult = testSHA512\_SA(result, Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 1, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало второго блока кода

IWriter writer = new FileWriter("test2.txt"); // Создаем экземпляр FileWriter для записи в файл test2.txt

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecSHA512(rsa); // Создаем экземпляр DecSHA512 для хэширования с использованием rsa

string? result = hash.Save("BBBBCCCCC"); // Хэшируем строку и сохраняем результат

StreamReader reader = new StreamReader(result); // Создаем StreamReader для чтения из результата

string? message = reader.ReadLine(); // Читаем первую строку из результата

reader.Close(); // Закрываем reader

bool testresult = testSHA512\_SA(message, Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 2, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало третьего блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecMD5(rsa); // Создаем экземпляр DecMD5 для хэширования с использованием rsa

string? result = hash.Save("AAAAACCCCBBBBB"); // Хэшируем строку и сохраняем результат

bool testresult = testMD5\_SA(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 3, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало четвертого блока кода

IWriter writer = new FileWriter("test4.txt"); // Создаем экземпляр FileWriter для записи в файл test4.txt

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecMD5(rsa); // Создаем экземпляр DecMD5 для хэширования с использованием rsa

string? result = hash.Save("AAAAABBBBDDDDDCCCCC"); // Хэшируем строку и сохраняем результат

StreamReader reader = new StreamReader(result); // Создаем StreamReader для чтения из результата

string? message = reader.ReadLine(); // Читаем первую строку результата

bool testresult = testMD5\_SA(message, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 4, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало пятого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter hash = new DecSHA512(writer); // Создаем экземпляр DecSHA512 для хэширования

string? result = hash.Save("AAAAAEEEEEBBBBCCCCC"); // Хэшируем строку и сохраняем результат

bool testresult = testSHA512(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 5, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало шестого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter hash = new DecMD5(writer); // Создаем экземпляр DecMD5 для хэширования

string? result = hash.Save("AAAAABBBBCCCCCHHHH"); // Хэшируем строку и сохраняем результат

bool testresult = testMD5(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 6, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало седьмого блока кода

IWriter writer = new FileWriter(); // Создаем экземпляр FileWriter без указания имени файла

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecSHA512(rsa); // Создаем экземпляр DecSHA512 для хэширования с использованием rsa

string? result = hash.Save("AAAAABBBBDDDDDCCCCC"); // Хэшируем строку и сохраняем результат

StreamReader reader = new StreamReader(result); // Создаем StreamReader для чтения из результата

string? message = reader.ReadLine(); // Читаем первую строку результата

bool testresult = testSHA512\_SA(message, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 7, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

// ------ НЕ УСПЕШНО --------------------------------------------------

{ // Начало восьмого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter hash = new DecSHA512(writer); // Создаем экземпляр DecSHA512 для хэширования

string? result = hash.Save("AAAAABBBBCCCCC"); // Хэшируем строку и сохраняем результат

bool testresult = testMD5(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования с использованием MD5

Console.WriteLine(string.Format("тест {0} - {1} ", 8, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало девятого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter hash = new DecSHA512(writer); // Создаем экземпляр DecSHA512 для хэширования

string? result = hash.Save(string.Format("{0}{1}{2}", "AAAAA", Lec5Lib.Constant.Delimiter, "BBBBCCCCC")); // Хэшируем строку с разделителем

bool testresult = testSHA512(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 9, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало десятого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter hash = new DecMD5(writer); // Создаем экземпляр DecMD5 для хэширования

string? result = hash.Save(string.Format("{0}{1}{2}", "AAAAA", Lec5Lib.Constant.Delimiter, "BBBBCCCCC")); // Хэшируем строку с разделителем

bool testresult = testSHA512(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования с использованием SHA-512

Console.WriteLine(string.Format("тест {0} - {1} ", 10, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало одиннадцатого блока кода

IWriter writer = new StrWriter(); // Создаем экземпляр StrWriter для записи в строку

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecMD5(rsa); // Создаем экземпляр DecMD5 для хэширования с использованием rsa

string? result = hash.Save("HHHAAAAABBBBCCCCC"); // Хэшируем строку и сохраняем результат

bool testresult = testSHA512\_SA(result, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 11, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало двенадцатого блока кода

IWriter writer = new FileWriter("test12.txt"); // Создаем экземпляр FileWriter для записи в файл test12.txt

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecSHA512(rsa); // Создаем экземпляр DecSHA512 для хэширования с использованием rsa

string? result = hash.Save("BBBBCCCCC"); // Хэшируем строку и сохраняем результат

StreamReader reader = new StreamReader(result); // Создаем StreamReader для чтения из результата

string? message = reader.ReadLine(); // Читаем первую строку результата

bool testresult = testSHA512\_SA("BBBBXCCCC", Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования с неправильным значением

Console.WriteLine(string.Format("тест {0} - {1} ", 12, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

{ // Начало тринадцатого блока кода

IWriter writer = new FileWriter("test13.txt"); // Создаем экземпляр FileWriter для записи в файл test13.txt

IWriter rsa = new DecRSA(writer); // Создаем экземпляр DecRSA для RSA шифрования с использованием writer

IWriter hash = new DecMD5(rsa); // Создаем экземпляр DecMD5 для хэширования с использованием rsa

string? result = hash.Save("BBBBCCCCC"); // Хэшируем строку и сохраняем результат

StreamReader reader = new StreamReader(result); // Создаем StreamReader для чтения из результата

string? message = reader.ReadLine(); // Читаем первую строку результата

bool testresult = testSHA512\_SA(message, Lec5Lib.Constant.Delimiter); // Проверяем результат хэширования

Console.WriteLine(string.Format("тест {0} - {1} ", 13, testresult ? "успешно" : "НЕ успешно")); // Выводим результат теста

}

}

// Метод для тестирования хэширования с использованием SHA-512

static bool testSHA512(string? message, char delimiter = '\uffff')

{

bool rc = false; // Инициализируем результат

try // Пытаемся выполнить код

{

message = message ?? string.Empty; // Заменяем null на пустую строку

int k1 = message.IndexOf(delimiter); // Находим индекс первого разделителя

if (k1 == -1) throw new Exception("delimiter not found"); // Если разделитель не найден, выбрасываем исключение

string xs = message.Substring(0, k1); // Извлекаем исходное сообщение

string xsbhcs = message.Substring(k1 + 1); // Извлекаем хэш

byte[] xsbhc = Convert.FromBase64String(xsbhcs); // Декодируем хэш из Base64

rc = eqSHA512(xs, xsbhc); // Сравниваем хэш с вычисленным значением

}

catch (Exception ex) // Обрабатываем исключения

{

Console.WriteLine(string.Format("testSHA512: {0}", ex.Message)); // Выводим сообщение об ошибке

}

return rc; // Возвращаем результат

}

// Метод для тестирования хэширования с использованием MD5

static bool testMD5(string? message, char delimiter = '\uffff')

{

bool rc = false; // Инициализируем результат

try // Пытаемся выполнить код

{

message = message ?? string.Empty; // Заменяем null на пустую строку

int k1 = message.IndexOf(delimiter); // Находим индекс первого разделителя

if (k1 == -1) throw new Exception("delimiter not found"); // Если разделитель не найден, выбрасываем исключение

string xs = message.Substring(0, k1); // Извлекаем исходное сообщение

string xsbhcs = message.Substring(k1 + 1); // Извлекаем хэш

byte[] xsbhc = Convert.FromBase64String(xsbhcs); // Декодируем хэш из Base64

rc = eqMD5(xs, xsbhc); // Сравниваем хэш с вычисленным значением

}

catch (Exception ex) // Обрабатываем исключения

{

Console.WriteLine(string.Format("testMD5: {0}", ex.Message)); // Выводим сообщение об ошибке

}

return rc; // Возвращаем результат

}

// Метод для тестирования хэширования с использованием SHA-512 с RSA

static bool testSHA512\_SA(string? message, char delimiter = '\uffff')

{

bool rc = false; // Инициализируем результат

try // Пытаемся выполнить код

{

message = message ?? string.Empty; // Заменяем null на пустую строку

int k1 = message.IndexOf(delimiter); // Находим индекс первого разделителя

int k2 = message.IndexOf(delimiter, k1 + 1); // Находим индекс второго разделителя

if (k1 == -1 || k2 == -1) throw new Exception("delimiter not found"); // Проверяем, найдены ли разделители

string xs = message.Substring(0, k1); // Извлекаем исходное сообщение

string xsbhcs = message.Substring(k1 + 1, k2 - k1 - 1); // Извлекаем хэш

string xparam = message.Substring(k2 + 1); // Извлекаем параметры RSA

byte[] xsbh = deRSA(xsbhcs, xparam); // Дешифруем хэш

rc = eqSHA512(xs, xsbh); // Сравниваем хэш с вычисленным значением

}

catch (Exception ex) // Обрабатываем исключения

{

Console.WriteLine(string.Format("testSHA512\_SA: {0}", ex.Message)); // Выводим сообщение об ошибке

}

return rc; // Возвращаем результат

}

// Метод для тестирования хэширования с использованием MD5 с RSA

static bool testMD5\_SA(string? message, char delimiter = '\uffff')

{

bool rc = false; // Инициализируем результат

try // Пытаемся выполнить код

{

message = message ?? string.Empty; // Заменяем null на пустую строку

int k1 = message.IndexOf(delimiter); // Находим индекс первого разделителя

int k2 = message.IndexOf(delimiter, k1 + 1); // Находим индекс второго разделителя

if (k1 == -1 || k2 == -1) throw new Exception("delimiter not found"); // Проверяем, найдены ли разделители

string xs = message.Substring(0, k1); // Извлекаем исходное сообщение

string xsbhcs = message.Substring(k1 + 1, k2 - k1 - 1); // Извлекаем хэш

string xparam = message.Substring(k2 + 1); // Извлекаем параметры RSA

byte[] xsbh = deRSA(xsbhcs, xparam); // Дешифруем хэш

rc = eqMD5(xs, xsbh); // Сравниваем хэш с вычисленным значением

}

catch (Exception ex) // Обрабатываем исключения

{

Console.WriteLine(string.Format("testMD5\_SA: {0}", ex.Message)); // Выводим сообщение об ошибке

}

return rc; // Возвращаем результат

}

// Метод для дешифрования с использованием RSA

static byte[] deRSA(string sbhcs, string xmlparam)

{

byte[] rc = { }; // Инициализируем массив для результата

try // Пытаемся выполнить код

{

byte[] xsbhc = Convert.FromBase64String(sbhcs); // Декодируем хэш из Base64

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider()) // Создаем экземпляр RSA

{

rsa.FromXmlString(xmlparam); // Загружаем параметры RSA из XML

rc = rsa.Decrypt(xsbhc, false); // Дешифруем х

Сonstant.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public static class Constant // Определяем статический класс Constant для хранения констант

{

public const char Delimiter = '\uffff'; // Определяем константу Delimiter, используемую как разделитель (символ U+FFFF)

public const string FileName = "DefaultName.txt"; // Определяем константу FileName, содержащую имя файла по умолчанию

}

}

DecMD5.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Security.Cryptography; // Подключаем пространство имен для криптографических функций

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public class DecMD5 : Decorator // Определяем класс DecMD5, который наследует от класса Decorator

{

// Конструктор класса, принимает IWriter и передает его в базовый класс

public DecMD5(IWriter writer) : base(writer) { }

// Переопределяем метод Save для сохранения сообщения с его хэшированием

public override string? Save(string? message)

{

if (message != null) // Проверяем, что сообщение не равно null

{

// Вычисляем хэш-код сообщения с помощью MD5

byte[] hash = MD5.Create().ComputeHash(Encoding.UTF8.GetBytes(message));

// Преобразуем хэш-код в строку и добавляем его к сообщению

string hashedMessage = message + Constant.Delimiter + Convert.ToBase64String(hash);

// Сохраняем зашифрованное сообщение с помощью декорируемого объекта-писателя

return writer?.Save(hashedMessage); // Возвращаем результат сохранения

}

return null; // Если сообщение равно null, возвращаем null

}

}

}

Decorator.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public abstract class Decorator : IWriter // Определяем абстрактный класс Decorator, реализующий интерфейс IWriter

{

protected IWriter? writer; // Защищенное поле для хранения ссылки на объект IWriter

// Конструктор класса, принимающий объект IWriter и инициализирующий поле writer

public Decorator(IWriter? writer)

{

this.writer = writer; // Сохраняем переданный объект в поле writer

}

// Переопределяем метод Save, который сохраняет сообщение, используя декорируемый объект writer

public virtual string? Save(string? message)

{

return writer?.Save(message); // Вызываем метод Save у объекта writer и возвращаем результат

}

}

}

DecRSA.cs

using Newtonsoft.Json; // Подключаем библиотеку Newtonsoft.Json для работы с JSON

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.ComponentModel; // Подключаем пространство имен для работы с компонентами

using System.Linq; // Подключаем пространство имен для LINQ

using System.Security.Cryptography; // Подключаем пространство имен для криптографических функций

using System.Security.Cryptography.X509Certificates; // Подключаем пространство имен для работы с сертификатами

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Text.RegularExpressions; // Подключаем пространство имен для работы с регулярными выражениями

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public class DecRSA : Decorator // Определяем класс DecRSA, который наследует от класса Decorator

{

// Конструктор класса, принимает IWriter и передает его в базовый класс

public DecRSA(IWriter writer) : base(writer) { }

// Переопределяем метод Save для сохранения сообщения с его шифрованием

public override string? Save(string message)

{

// Генерация ключей RSA

string publicKeyXml; // Переменная для хранения публичного ключа в формате XML

byte[] encyptedData; // Переменная для хранения зашифрованных данных

message = message ?? string.Empty; // Заменяем null на пустую строку

// Находим индекс разделителя

int k1 = message.IndexOf(Constant.Delimiter);

if (k1 == -1) throw new Exception("delimiter not found"); // Выбрасываем исключение, если разделитель не найден

// Извлекаем оригинальное сообщение и хэш

string xs = message.Substring(0, k1); // Оригинальное сообщение

string xsbhcs = message.Substring(k1 + 1); // Хэш-код сообщения

// Вычисление хеша

byte[] xsbhc = Convert.FromBase64String(xsbhcs); // Декодируем хэш из Base64

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider()) // Создаем экземпляр RSA

{

publicKeyXml = rsa.ToXmlString(true); // Получаем публичный ключ в XML формате

rsa.ImportParameters(rsa.ExportParameters(false)); // Импортируем параметры RSA

encyptedData = rsa.Encrypt(xsbhc, false); // Шифруем данные

}

// Формируем результат для записи

string result = $"{xs}{Lec5Lib.Constant.Delimiter}{Convert.ToBase64String(encyptedData)}{Lec5Lib.Constant.Delimiter}{publicKeyXml}";

return writer.Save(result); // Сохраняем результат с помощью декорируемого объекта-писателя

}

}

}

DecSHA512.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Security.Cryptography; // Подключаем пространство имен для криптографических функций

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public class DecSHA512 : Decorator // Определяем класс DecSHA512, который наследует от класса Decorator

{

// Конструктор класса, принимает IWriter и передает его в базовый класс

public DecSHA512(IWriter writer) : base(writer) { }

// Переопределяем метод Save для сохранения сообщения с его хэшированием

public override string? Save(string? message) // Формирование сообщения

{

if (message != null) // Проверяем, что сообщение не равно null

{

// Вычисляем хэш-код сообщения с помощью SHA-512

byte[] hash = SHA512.Create().ComputeHash(Encoding.UTF8.GetBytes(message));

// Преобразуем хэш-код в строку и добавляем его к сообщению

string hashedMessage = message + Constant.Delimiter + Convert.ToBase64String(hash);

// Сохраняем зашифрованное сообщение с помощью декорируемого объекта-писателя

return writer?.Save(hashedMessage); // Возвращаем результат сохранения

}

return null; // Если сообщение равно null, возвращаем null

}

}

}

FileWriter.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Data; // Подключаем пространство имен для работы с данными

using System.Linq; // Подключаем пространство имен для LINQ

using System.Reflection.Metadata; // Подключаем пространство имен для работы с метаданными

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

using System.IO; // Подключаем пространство имен для работы с файловой системой

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public class FileWriter : IWriter // Определяем класс FileWriter, реализующий интерфейс IWriter

{

string filename = Constant.FileName; // Имя файла по умолчанию

public string FileName { get { return this.filename; } } // Свойство для получения имени файла

// Конструктор класса, принимает имя файла или использует значение по умолчанию

public FileWriter(string? filename = null)

{

this.filename = filename ?? Constant.FileName; // Если имя файла не указано, используем значение по умолчанию

}

// Метод для сохранения текста в файл

public string? Save(string? text)

{

try // Пытаемся выполнить код

{

using StreamWriter writer = new StreamWriter(filename); // Создаем StreamWriter для записи в файл

writer.WriteLine(text); // Записываем текст в файл

}

catch (Exception) // Обрабатываем исключения

{

Console.WriteLine("Что-то пошло не так "); // Выводим сообщение об ошибке

return null; // Возвращаем null в случае ошибки

}

return this.filename; // Возвращаем имя файла после успешного сохранения

}

}

}

IWriter.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public interface IWriter // Определяем интерфейс IWriter

{

// Метод для сохранения сообщения, возвращает null в случае ошибки

string? Save(string? message);

}

}

StrWriter.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections; // Подключаем пространство имен для коллекций

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Security.Cryptography; // Подключаем пространство имен для криптографических функций

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lec5Lib // Определяем пространство имен Lec5Lib

{

public class StrWriter : IWriter // Определяем класс StrWriter, реализующий интерфейс IWriter

{

// Метод для сохранения сообщения

public string? Save(string? message)

{

return message; // Возвращаем полученное сообщение (без изменений)

}

}

}

ОТВЕТЫ НА ВОПРОСЫ :

### 6. Поясните суть паттерна Decorator, что дает его применение.

Паттерн \*\*Decorator\*\* позволяет динамически добавлять новые обязанности (или функциональность) объектам, не меняя их структуру. Это достигается путем обертывания объекта в новый класс, который реализует тот же интерфейс, что и оригинальный объект.

\*\*Преимущества применения:\*\*

- \*\*Гибкость\*\*: Позволяет добавлять или изменять функциональность объектов на лету.

- \*\*Избежание наследования\*\*: Снижает необходимость создавать множество подклассов для расширения функциональности.

- \*\*Композиция\*\*: Объекты можно комбинировать, создавая сложные поведения, что упрощает код и улучшает его читаемость.

### 7. Что необходимо добавить в библиотеку Lab5Lib, чтобы добавить новый метод хеширования?

Чтобы добавить новый метод хеширования в библиотеку Lab5Lib, необходимо:

1. \*\*Создать новый класс декоратора\*\* (например, `DecNewHash`) для реализации нового метода хеширования.

2. \*\*Реализовать интерфейс `IWriter`\*\*, чтобы сохранить результат хеширования.

3. \*\*Использовать соответствующий алгоритм хеширования\*\* (например, SHA-256 или другой), импортировав необходимые пространства имен.

4. \*\*Определить логику для обработки входного сообщения\*\* и возврата хэшированного значения.

### 8. Что необходимо добавить в библиотеку Lab5Lib, чтобы добавить новый метод шифрования?

Чтобы добавить новый метод шифрования в библиотеку Lab5Lib, необходимо:

1. \*\*Создать новый класс декоратора\*\* (например, `DecNewEncryption`) для реализации нового метода шифрования.

2. \*\*Реализовать интерфейс `IWriter`\*\*, чтобы сохранить зашифрованные данные.

3. \*\*Выбрать алгоритм шифрования\*\* (например, AES, DES и т.д.), и импортировать необходимые пространства имен.

4. \*\*Определить логику для обработки входного сообщения\*\* и возврата зашифрованных данных.

### 9. Что необходимо добавить в библиотеку Lab5Lib, чтобы формируемое сообщение было в формате протокола http/1.1?

Чтобы формируемое сообщение соответствовало формату протокола HTTP/1.1, необходимо:

1. \*\*Создать новый класс или метод\*\* для форматирования сообщения в соответствии с HTTP-стандартами.

2. \*\*Определить заголовки\*\* (например, `Content-Type`, `Content-Length`) и сформировать структуру сообщения, включая статусный код и тело ответа.

3. \*\*Обновить методы сохранения\*\* (например, в классе `FileWriter` или `StrWriter`), чтобы они могли принимать и обрабатывать сообщения в формате HTTP.

4. \*\*Добавить обработку ошибок и статусные коды\*\*, чтобы обеспечить соответствие протоколу.

Если есть дополнительные вопросы или требуется больше деталей, дайте знать!

6. Паттерн Decorator - это структурный паттерн проектирования, который позволяет динамически добавлять объектам новые функциональные возможности путем помещения их в объекты-обертки. Таким образом, Decorator позволяет расширять функциональность объекта без изменения его класса и без влияния на поведение других экземпляров того же класса. Это достигается путем создания цепочки объектов-оберток, каждый из которых добавляет свою уникальную функциональность к исходному объекту.

7. Чтобы добавить новый метод хеширования в библиотеку Lab5Lib, необходимо создать новый класс, реализующий интерфейс, определяющий методы для работы с хешированием. Этот класс может содержать реализацию любого нового алгоритма хеширования, который вы хотите добавить. Затем необходимо обновить класс, который будет использовать этот новый класс для хеширования.

8. Чтобы формируемое сообщение было в формате протокола http/1.1 в библиотеку Lab5Lib, необходимо создать класс, который будет отвечать за форматирование и отправку сообщений по протоколу http/1.1. Этот класс должен реализовывать все необходимые методы для формирования правильного формата сообщения, такие как методы для добавления заголовков, тела сообщения и т.д. Затем необходимо обновить класс, который будет использовать этот новый класс для формирования и отправки сообщений по протоколу http/1.1.