Факультет компьютерных технологий

Лабораторная работа №12

«Разработка программ с использованием управляющих инструкций»

Вариант №10

Выполнил: студент гр. 981063 Ефименко Павел Викторович

Проверил: Яковлев А.В.

Минск 2021

Задание №1 Шаблон проектирования - фабрика

Условие:

Создать фабрику алгоритмов шифрования.

Код:

using System;

using System.Collections.Generic;

using Laba\_4.Cryptos;

namespace Laba\_4

{

internal class Program

{

private static void Main()

{

var message = "колобок";

ICryptAlgorithmFactory factory = new CryptAlgorithmFactory();

var algorithms = new List<ICrypt>

{

factory.CreateMagicSquare(),

factory.CreatePolibSquare(),

factory.CreateWheatstoneSquare()

};

foreach (var algorithm in algorithms)

{

DemoCryptAlgorithm(algorithm, message);

}

}

private static void DemoCryptAlgorithm(ICrypt algorithm, string message)

{

var encryptMessage = algorithm.Crypt(message);

var decryptedMessage = algorithm.Decrypt(encryptMessage);

Console.WriteLine(encryptMessage);

Console.WriteLine(decryptedMessage);

Console.WriteLine();

}

}

}

using Laba\_4.Cryptos;

namespace Laba\_4

{

public interface ICryptAlgorithmFactory

{

ICrypt CreateMagicSquare();

ICrypt CreatePolibSquare();

ICrypt CreateWheatstoneSquare();

}

}

using Laba\_4.Cryptos;

namespace Laba\_4

{

internal class CryptAlgorithmFactory : ICryptAlgorithmFactory

{

public ICrypt CreateMagicSquare()

{

return new MagicSquare();

}

public ICrypt CreatePolibSquare()

{

return new PolibSquare();

}

public ICrypt CreateWheatstoneSquare()

{

return new WheatstoneSquare();

}

}

}

using System;

namespace Laba\_4.Utils

{

public static class MatrixUtil

{

public static string ConvertToString<T>(this T[,] matrix)

{

var result = string.Empty;

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

result += matrix[i, j].ToString();

}

}

return result;

}

public static T[,] ConvertToMatrix<T>(this T[] arr, int m, int n)

{

if (arr.Length != m \* n)

{

throw new ArgumentNullException();

}

T[,] matrix = new T[m, n];

Buffer.BlockCopy(arr, 0, matrix, 0, arr.Length \* sizeof(char));

return matrix;

}

}

}

namespace Laba\_4.Cryptos

{

public interface ICrypt

{

string Crypt(string text);

string Decrypt(string encryptText);

}

}

using System;

using Laba\_4.Utils;

namespace Laba\_4.Cryptos

{

internal class MagicSquare : ICrypt

{

private const int DefaultSquareSize = 4;

public string Crypt(string text)

{

var crypted = this.EncryptionMagicSquare(text);

return crypted.ConvertToString();

}

public string Decrypt(string text)

{

var cryptedMessage = text.ToCharArray().ConvertToMatrix(DefaultSquareSize, DefaultSquareSize);

return this.DecipherMagicSquare(cryptedMessage);

}

private int[,] defaultSquare = new int[DefaultSquareSize, DefaultSquareSize] { { 16, 3, 2, 13 }, { 9, 6, 7, 12 }, { 5, 10, 11, 8 }, { 4, 15, 14, 1 } };

private char[,] EncryptionMagicSquare(string text)

{

int index = 0;

char[,] encryptedText = new char[4, 4] { { ' ', ' ', ' ', ' ' }, { ' ', ' ', ' ', ' ' }, { ' ', ' ', ' ', ' ' }, { ' ', ' ', ' ', ' ' } };

while (index != text.Length)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 4; j++)

{

if ((defaultSquare[i, j] == index + 1) && (index != text.Length))

{

encryptedText[i, j] = text[index];

index++;

}

}

}

}

return encryptedText;

}

private string DecipherMagicSquare(char[,] text)

{

int index = 0;

string decipheredText = null;

while (index != text.Length)

{

for (int i = 0; i < text.GetLength(0); i++)

{

for (int j = 0; j < text.GetLength(1); j++)

{

if ((defaultSquare[i, j] == index + 1) && (index != text.Length))

{

decipheredText += Convert.ToString(text[i, j]);

index++;

}

}

}

}

return decipheredText;

}

}

}

namespace Laba\_4.Cryptos

{

/// <summary>

/// Квадрат Полибия

/// </summary>

internal class PolibSquare : ICrypt

{

/// <summary>

/// Таблица шифрования

/// </summary>

private readonly char[,] arr;

private const int ArrLenght = 6;

public PolibSquare()

{

arr = new char[ArrLenght, ArrLenght];

FillArrayByAlphabet();

}

public string Crypt(string text)

{

return this.Encrypt(text);

}

/// <summary>

/// Заполнить таблицу шифрования по алфавиту

/// </summary>

public void FillArrayByAlphabet()

{

char letter = 'а';

for (int i = 0; i < ArrLenght; i++)

{

for (int j = 0; j < ArrLenght; j++)

{

arr[i, j] = letter;

if (letter == 'я')

break;

letter++;

}

}

}

/// <summary>

/// Первый вариант шифрования

/// Находим буву в таблице и вместо неё записываем нижнюю от неё в этом же столбце

/// Если буква была в нижней строчке, берём из первой

/// </summary>

/// <param name="str"></param>

/// <returns></returns>

public string Encrypt(string str)

{

string result = null;

if (!CheckString(str))

return "В строке присутсвуют запрещенные символы";

foreach (var letter in str.ToLower()) // берём букву из строки

{

if (letter == ' ') // пробел просто добавляем в резалт строку

{

result += letter;

continue;

}

for (int i = 0; i < ArrLenght; i++)

{

for (int j = 0; j < ArrLenght; j++)

{

if (letter == arr[i, j]) // находим нужную букву в таблице

{

if (i == ArrLenght - 1 || arr[i + 1, j] == '\0') // если буква из последней строки или под ней пусто берём букву из первой строки

result += arr[0, j];

else

result += arr[i + 1, j]; // иначе просто из нижней строки

}

}

}

}

return result;

}

public string Decrypt(string str) // все тоже самое что в Encrypt только наоборот

{

string result = null;

if (!CheckString(str))

return "В строке присутсвуют запрещенные символы";

foreach (var letter in str.ToLower())

{

if (letter == ' ')

{

result += letter;

continue;

}

for (int i = 0; i < ArrLenght; i++)

{

for (int j = 0; j < ArrLenght; j++)

{

if (letter == arr[i, j])

{

if (i == 0)

{

if (arr[ArrLenght - 1, j] == '\0')

result += arr[ArrLenght - 2, j];

else result += arr[ArrLenght - 1, j];

}

else

result += arr[i - 1, j];

}

}

}

}

return result;

}

private bool CheckString(string str) // проверить строку на правильность ввода

{

foreach (var letter in str.ToLower())

{

if (!(letter >= 'а' && letter <= 'я' || letter == ' '))

return false;

}

return true;

}

}

}

namespace Laba\_4.Cryptos

{

internal class WheatstoneSquare : ICrypt

{

private readonly char[,] \_firstMatrix = new char[7, 5]

{

{'ж', 'щ', 'н', 'ю', 'р'},

{'и', 'т', 'ь', 'ц', 'б'},

{'я', 'м', 'е', '.', 'с'},

{'в', 'ы', 'п', 'ч', ' '},

{'й', 'д', 'у', 'о', 'к'},

{'з', 'э', 'ф', 'г', 'ш'},

{'х', 'а', ',', 'л', 'ъ'}

};

private readonly char[,] \_secondMatrix = new char[7, 5]

{

{'и', 'ч', 'г', 'я', 'т'},

{',', 'ж', 'м', 'ь', 'о'},

{'з', 'ю', 'р', 'в', 'щ'},

{'ц', 'й', 'п', 'е', 'л'},

{'ъ', 'а', 'н', '.', 'х'},

{'э', 'к', 'с', 'ш', 'д'},

{'б', 'ф', 'у', 'ы', ' '}

};

public string Crypt(string text)

{

return this.Encrypt(text);

}

public string Encrypt(string text)

{

text = text.ToLower();

string result\_text = "";

if (text.Length % 2 != 0)

{

text += ' ';

}

int length = text.Length / 2;

int k = 0;

char[,] bigram = new char[length, 2];

char[,] kripto\_bigram = new char[length, 2];

for (int i = 0; i < length; i++)

{

for (int j = 0; j < 2; j++)

{

bigram[i, j] = text[k];

k++;

}

}

int step = 0;

while (step < length)

{

var cortege1 = FindIndexes(bigram[step, 0], \_firstMatrix);

var cortege2 = FindIndexes(bigram[step, 1], \_secondMatrix);

kripto\_bigram[step, 0] = \_secondMatrix[(int)cortege1.i, (int)cortege2.j];

kripto\_bigram[step, 1] = \_firstMatrix[(int)cortege2.i, (int)cortege1.j];

step++;

}

for (int i = 0; i < length; i++)

{

for (int j = 0; j < 2; j++)

{

result\_text += kripto\_bigram[i, j].ToString();

}

}

return result\_text;

}

public string Decrypt(string text)

{

string result\_text = "";

int length = text.Length / 2;

int k = 0;

char[,] bigram = new char[length, 2];

char[,] kripto\_bigram = new char[length, 2];

for (int i = 0; i < length; i++)

{

for (int j = 0; j < 2; j++)

{

bigram[i, j] = text[k];

k++;

}

}

int step = 0;

while (step < length)

{

var cortege1 = FindIndexes(bigram[step, 0], \_secondMatrix);

var cortege2 = FindIndexes(bigram[step, 1], \_firstMatrix);

kripto\_bigram[step, 0] = \_firstMatrix[(int)cortege1.i, (int)cortege2.j];

kripto\_bigram[step, 1] = \_secondMatrix[(int)cortege2.i, (int)cortege1.j];

step++;

}

for (int i = 0; i < length; i++)

{

for (int j = 0; j < 2; j++)

{

result\_text += kripto\_bigram[i, j].ToString();

}

}

return result\_text;

}

private (int? i, int? j) FindIndexes(char symbol, char[,] matrix)

{

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 5; j++)

{

if (symbol == matrix[i, j])

{

return (i, j);

}

}

}

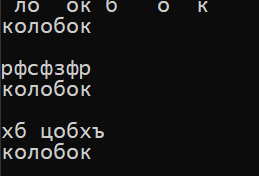
return (i: null, j: null);

}

}

}

Результат:



Вывод

В ходе лабораторной работы были отработаны навыки разработки алгоритма на основе шаблонов проектирования, таких как фабрика и фабричный метод.