МИНИСТЕРСТВО ОБРАЗОВАНИЯ, КУЛЬТУРЫ И ИССЛЕДОВАНИЯ РЕСПУБЛИКИ МОЛДОВА

ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ МОЛДОВЫ

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ, ИНФОРМАТИКИ И МИКРОЭЛЕКТРОНИКИ

ДЕПАРТАМЕНТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И АВТОМАТИКИ

ОТЧЕТ

По лабораторной работе №1

по дисциплине:

Объектно-ориентированный анализ, проектирование и программирование

Тема: Основные принципы объектно-ориентированной парадигмы

Выполнил: студент группы TI-154 Лукаш Сергей

Проверил: Пекарь Михаил

**Цель работы**:

Изучение основных принципов объектно-ориентированной парадигмы: абстракция, наследование, полиморфизм и инкапсуляция.

**Ход работы**

Исходный код проекта можно найти по следующему адресу:

<https://github.com/SergeyLucash/APPOO/tree/master/Lab1/Crypto>

В процессе изучения были выведены 4 принципа, на которых базируется вся объектно-ориентированная парадигма. Для полного охвата всех принципов, был создан небольшой проект, в котором и реализуются примеры всех принципов.

**Абстракция**

Абстрагирование означает выделение значимой информации и исключение из рассмотрения незначимой. В ООП рассматривают лишь абстракцию данных, подразумевая набор значимых характеристик объекта, доступный остальной программе.

Примером абстракции в проекте является следующий абстрактный класс:

public abstract class AbstractCrypter

{

public readonly bool UsesCodeword;

public readonly bool UsesDictionary;

protected AbstractCrypter(bool usesCodeword, bool usesDictionary)

{

UsesCodeword = usesCodeword;

UsesDictionary = usesDictionary;

}

public abstract string Encrypt(string textToEncrypt, AdditionalData data = null);

public abstract string Decrypt(string encryptedText, AdditionalData data = null);

}

Он представляет собой абстрактный шифровщик, объединяющий в себе набор 2 основных для каждого шифровщика параметров: UsesCodeword – как флаг использования шифровальщиком кодового слова, и Uses Dictionary – как флаг использования словаря. Данный класс также определяет две основные функции шифровщика: Encrypt – метод для шифрования, и Decrypt – метод для дешифровки.

**Наследование**

Наследование — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Наследование в проекте представляется двумя классами: VisinerCrypter и CaesarCrypter. Также есть наследование от класса AdditionalData, но для примера будут рассматриваться только вышеупомянутые классы. Данные классы наследуются от AbstractCrypter. Каждый из классов в конструкторе задает свои параметры UsesCodeword и UsesDictionary, а также переопределяют методы Encrypt и Decrypt, унаследованные от AbstractCrypter:

public class VisinerCrypter : AbstractCrypter

{

public VisinerCrypter() : base(true, true) {}

public override string Encrypt(string text, AdditionalData data = null)

{

if (!(data is VisinerData vData))

{

throw new ArgumentException("Неверный параметр", nameof(data));

}

char[] encrypted = text.ToCharArray();

var codeWord = new string(vData.Codeword.ToCharArray());

var dictionary = vData.Dictionary;

if (vData.Codeword.Length != text.Length)

{

codeWord = ResizeCodeWord(codeWord, text.Length);

}

for (var i = 0; i < encrypted.Length; i++)

{

var index = (dictionary.IndexOf(text[i]) + dictionary.IndexOf(codeWord[i])) % dictionary.Length;

encrypted[i] = dictionary.AtIndex(index);

}

return new string(encrypted);

}

public override string Decrypt(string encryptedText, AdditionalData data = null)

{

if (!(data is VisinerData vData))

{

throw new ArgumentException("Неверный параметр", nameof(data));

}

char[] decrypted = encryptedText.ToCharArray();

var codeWord = new string(vData.Codeword.ToCharArray());

var dictionary = vData.Dictionary;

if (codeWord.Length != encryptedText.Length)

{

codeWord = ResizeCodeWord(codeWord, encryptedText.Length);

}

for (var i = 0; i < decrypted.Length; i++)

{

var index = (dictionary.IndexOf(encryptedText[i]) - dictionary.IndexOf(codeWord[i]) + dictionary.Length) % dictionary.Length;

decrypted[i] = dictionary.AtIndex(index);

}

return new string(decrypted);

}

private static string ResizeCodeWord(string codeWord, int size)

{

var repeats = size / codeWord.Length;

var resizedCodeWord = codeWord.Substring(0);

for (var i = 0; i < repeats; i++)

resizedCodeWord += codeWord;

return resizedCodeWord.Substring(0, size);

}

}

public class CaesarCrypter : AbstractCrypter

{

public CaesarCrypter() : base(false, false) {}

public override string Encrypt(string text, AdditionalData data = null)

{

char[] encrypted = text.ToCharArray();

for (var i = 0; i < encrypted.Length; i++)

{

encrypted[i] = (char) (encrypted[i] + 1);

}

return new string(encrypted);

}

public override string Decrypt(string encryptedText, AdditionalData data = null)

{

char[] decrypted = encryptedText.ToCharArray();

for (var i = 0; i < decrypted.Length; i++)

{

decrypted[i] = (char)(decrypted[i] - 1);

}

return new string(decrypted);

}

}

**Полиморфизм**

Полиморфизм подтипов, или просто полиморфизм, — свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. Полиморфизм также позволяет дочерним классам переопределять методы базового класса.

Примером полиморфизма в проекте является переопределение методов Encrypt и Decrypt в классах VisinerCrypter и CaesarCrypter, упомянутых ранее. Другим примером может быть использование объекта абстрактного класса AbstractCrypter \_currentCrypter, которому на самом деле присваиваются различные значения:

\_currentCrypter = new VisinerCrypter();

\_currentCrypter = new CaesarCrypter();

А использование методов класса AbstractCrypter на самом деле приводит к использованию к методам классов, на которые и указывает \_currentCrypter. Все, что знает система, что ей необходимо передать в параметры текст и какие-то дополнительные данные: \_currentCrypter.Encrypt(text, \_additionalData).

**Инкапсуляция**

Инкапсуляция — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Одни языки (например, С++, Java или Ruby) отождествляют инкапсуляцию с сокрытием – использованием модификаторов доступа для разграничения доступа к параметрам класса, но другие (Smalltalk, Eiffel, OCaml) различают эти понятия.

Для используемого языка C#, под инкапсуляцией понимается сокрытие параметров класса. Так, примеры разграничения доступа к полям и методам:

public class SymbolDictionary

{

private string \_content = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public string Content

{

get => \_content;

set { \_content = new string(value.Distinct().OrderBy(x => x).ToArray()); }

}

public int Length => Content.Length;

public bool Contains(char ch)

{

return Content.IndexOf(ch) >= 0;

}

public bool ContainsAll(string str)

{

return str.All(Contains);

}

public int IndexOf(char c)

{

var index = Content.IndexOf(c);

return index;

}

public string AbsentSymbols(string str)

{

var builder = new StringBuilder();

foreach (var ch in str)

{

if (!Contains(ch))

{

builder.Append(ch);

}

}

return builder.ToString();

}

public char AtIndex(int index)

{

return Content[index];

}

}

В данном классе поле \_content помечено как private. Это гарантирует, что пользователь извне не сможет изменить данный параметр. Для этого должны использоваться public методы и свойства класса.

Также, как пример сокрытия можно рассматривать следующие поля класса MainWindow:

private AbstractCrypter \_currentCrypter;

private AdditionalData \_additionalData;

private readonly SymbolDictionary \_dictionary;

**Результат работы**

Приведем скриншоты работы программы для шифра Цезаря: выбор метода (рис. 1), шифрование (рис. 2), дешифровка (рис. 3).

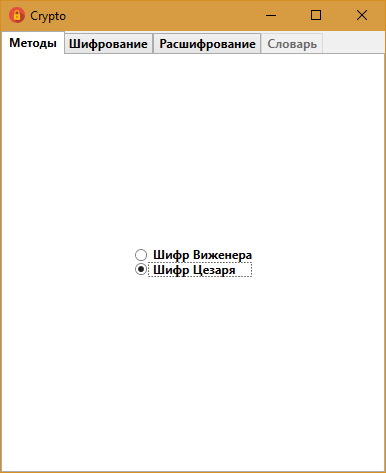


Рисунок 1 – Окно выбора метода шифрования

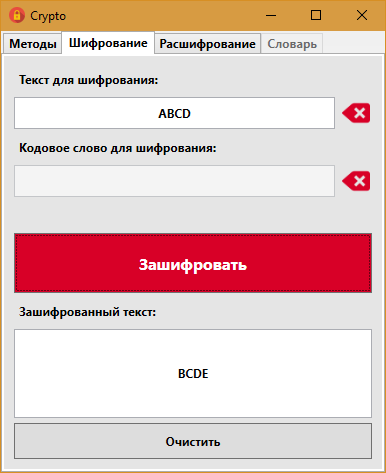


Рисунок 2 – Шифрование с помощью кода Цезаря

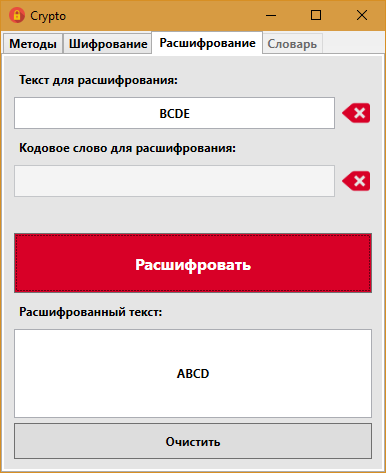


Рисунок 3 – Дешифровка с помощью кода Цезаря

**Вывод**

При выполнении данной лабораторной работы мы изучили основные термины и принципы, на которых базируется парадигма ООП: абстракция, наследование, полиморфизм, инкапсуляция. В процессе выполнения, мы убедились, что ООП предоставляет мощные инструменты для создания приложений, основанных на объектной модели.

Для примера использования всех данных принципов на практике, был создан проект, представляющий собой простейший шифровщик-дешифровщик текстовой информации. Он позволяет использовать два метода шифрования-дешифровки – это код Виженера и код Цезаря. В результате мы получили работающую программу, реализующую данный функционал, позволяющую изменять во время работы метод шифрования-дешифровки, а также в которой явно можно проследить наличие всех 4 принципов ООП.