МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

кафедра Информационные системы

Бариев Эмин Юсуфович

Институт информационных технологий и управления в технических системах

курс 4 группа ИС/б-16-2

09.03.02 Информационные системы и технологии (уровень бакалавриата)

ОТЧЕТ

по лабораторной работе №7

По дисциплине: «Рефакторинг программного обеспечения»

По теме: «Исследование способов применения порождающих паттернов проектирования при рефакторинге ПО»

|  |  |  |
| --- | --- | --- |
| Отметка о зачете | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_ |
|  |  | (дата) |
|  | Руководитель практикума |  |
| ст. преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Строганов В.А. |
| (должность) | (подпись) | (инициалы, фамилия) |

Севастополь 2019

**1 Цель работы**

Исследовать возможность использования порождающих паттернов проектирования. Получить практические навыки применения порождающих паттернов при объектно-ориентированном проектировании и рефакторинге ПО.

**2 Постановка задачи**

2.1 Изучить назначение и структуру паттерна ***Абстрактная фабрика*** (выполнить в ходе самостоятельной подготовки).

2.2 Применительно к программному продукту, выбранному для рефакторинга, проанализировать возможность использования паттерна ***Абстрактная фабрика***. Для этого построить диаграмму классов, на диаграмме классов выделить семейства взаимосвязанных и совместно используемых классов, которые должны инстанцироваться совместно и при этом инстанцирующий их клиент не должен быть привязан к конкретным именам классов (пример приведен в разделе 2.2.).

2.3 Выполнить перепроектирование системы, использовав паттерн ***Абстрактная фабрика***, изменения отобразить на диаграмме классов.

2.4 Сравнить полученные диаграммы классов, сделать выводы и целесообразности использования паттернов проектирования для данной системы.

2.5 На основе полученной UML-диаграммы модифицировать программный код, скомпилировать программу, выполнить ее тестирование и продемонстрировать ее работоспособность.

**3 Ход работы**

Диаграмма классов для созданной программы с использованием паттерна «Абстрактная фабрика».

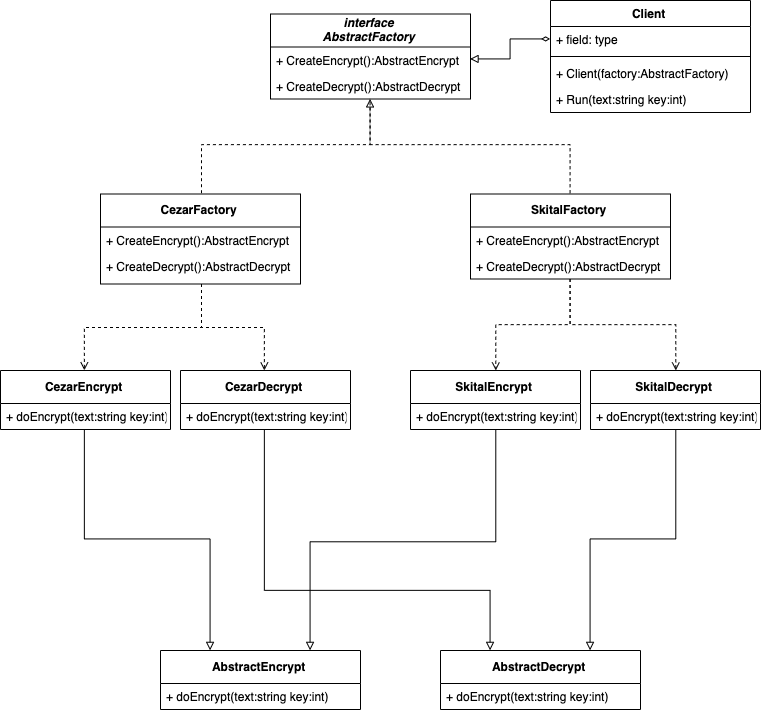


Рисунок 1 – Диаграмма классов с реализацией паттерна «Абстрактная фабрика»

Программный код на языке C#

using System;

namespace ref\_7

{

class Program

{

#region Abs classes

abstract class AbstractEncrypt

{

public abstract string doEncrypt(string text, int k);

}

abstract class AbstractDecrypt

{

public abstract string doDecrypt(string text, int k);

}

#endregion

class CezarEcrypt : AbstractEncrypt

{

public override string doEncrypt(string text, int k)

{

const string alfabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

//добавляем в алфавит маленькие буквы

var fullAlfabet = alfabet + alfabet.ToLower();

var letterQty = fullAlfabet.Length;

var retVal = "";

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

{

var c = text[i];

var index = fullAlfabet.IndexOf(c);

if (index < 0)

{

//если символ не найден, то добавляем его в неизменном виде

retVal += c.ToString();

}

else

{

var codeIndex = (letterQty + index + k) % letterQty;

retVal += fullAlfabet[codeIndex];

}

}

return retVal;

}

}

class SkitalEncrypt : AbstractEncrypt

{

public override string doEncrypt(string text, int k)

{

var t = text.Length % k;

if (t > 0)

{

//дополняем строку пробелами

text += new string(' ', k - t);

}

var column = text.Length / k;

var result = "";

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

{

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

{

result += text[i + column \* j].ToString();

}

}

return result;

}

}

class CezarDecrypt : AbstractDecrypt

{

public override string doDecrypt(string text, int k)

{

k = -k;

const string alfabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

//добавляем в алфавит маленькие буквы

var fullAlfabet = alfabet + alfabet.ToLower();

var letterQty = fullAlfabet.Length;

var retVal = "";

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

{

var c = text[i];

var index = fullAlfabet.IndexOf(c);

if (index < 0)

{

//если символ не найден, то добавляем его в неизменном виде

retVal += c.ToString();

}

else

{

var codeIndex = (letterQty + index + k) % letterQty;

retVal += fullAlfabet[codeIndex];

}

}

return retVal;

}

}

class SkitalDecrypt : AbstractDecrypt

{

public override string doDecrypt(string text, int k)

{

var column = text.Length / k;

var symbols = new char[text.Length];

int index = 0;

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

{

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

{

symbols[i + column \* j] = text[index];

index++;

}

}

return string.Join("", symbols);

}

}

#region Abstract Factory

abstract class AbstractFactory

{

public abstract AbstractEncrypt CreateEncrypt();

public abstract AbstractDecrypt CreateDecrypt();

}

#endregion

class CezarFactory : AbstractFactory

{

public override AbstractEncrypt CreateEncrypt()

{

return new CezarEcrypt();

}

public override AbstractDecrypt CreateDecrypt()

{

return new CezarDecrypt();

}

}

class SkitalFactory : AbstractFactory

{

public override AbstractEncrypt CreateEncrypt()

{

return new SkitalEncrypt();

}

public override AbstractDecrypt CreateDecrypt()

{

return new SkitalDecrypt();

}

}

class Client

{

private AbstractEncrypt encrypt;

private AbstractDecrypt decrypt;

public Client(AbstractFactory factory)

{

encrypt = factory.CreateEncrypt();

decrypt = factory.CreateDecrypt();

}

public void Run(string text, int key)

{

string result = encrypt.doEncrypt(text, key);

Console.WriteLine("Encrypted: " + result);

Console.WriteLine("Decrypted: " + decrypt.doDecrypt(result, key));

Console.WriteLine();

}

}

static void Main(string[] args)

{

Client client = null;

Console.WriteLine("Cezar chipher");

client = new Client(new CezarFactory());

client.Run("ПРИВЕТ МИР",2);

Console.WriteLine("Skitalа chipher");

client = new Client(new SkitalFactory());

client.Run("HELLO WORLD", 3);

}

}

}

**Выводы**

В ходе выполнения лабораторной работы был исследован и применен порождающих паттернов проектирования – «Абстрактная фабрика». Были получены практические навыки применения порождающих паттернов при объектно-ориентированном проектировании и рефакторинге ПО.