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 №5

По дисциплине: «Рефакторинг программного обеспечения»

По теме: «Исследование способов применения структурных 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 структурных паттернов проектирования. Получить практические навыки применения структурных паттернов при объектно-ориентированном проектировании и рефакторинге ПО.

**2 Постановка задачи**

3.1. Ознакомиться с основными преимуществами объектно- ориентированного проектирования на основе паттернов, изучить порядок проектирования с использованием паттернов. Изучить назначение и структуру паттерна *Адаптер* (выполнить в ходе самостоятельной подготовки).

3.2. Применительно к программному продукту, выбранному для рефакторинга, проанализировать возможность использования паттерна *Адаптер*. Для этого построить диаграмму классов, на диаграмме классов найти класс-клиент и адаптируемый класс, функциональностью которого должен воспользоваться клиент.

3.3. Выполнить перепроектирование системы, использовав паттерн *Адаптер*, изменения отобразить на диаграмме классов.

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

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

**3 Ход работы**

Даны два класса с разными интерфейсами: *IMatrix* и *MatrixMultMethods*. Класс *MatrixMultMethods* реализует умножение матриц, интерфейс клиента имеет метод calculate (для перемножения матриц), в который следует завернуть метод стороннего класса, чтобы клиент мог воспользоваться этой функцией и нам бы не пришлось писать «сложный» код перемножения матриц. Реализуем это с помощью паттерна «*Adapter*».

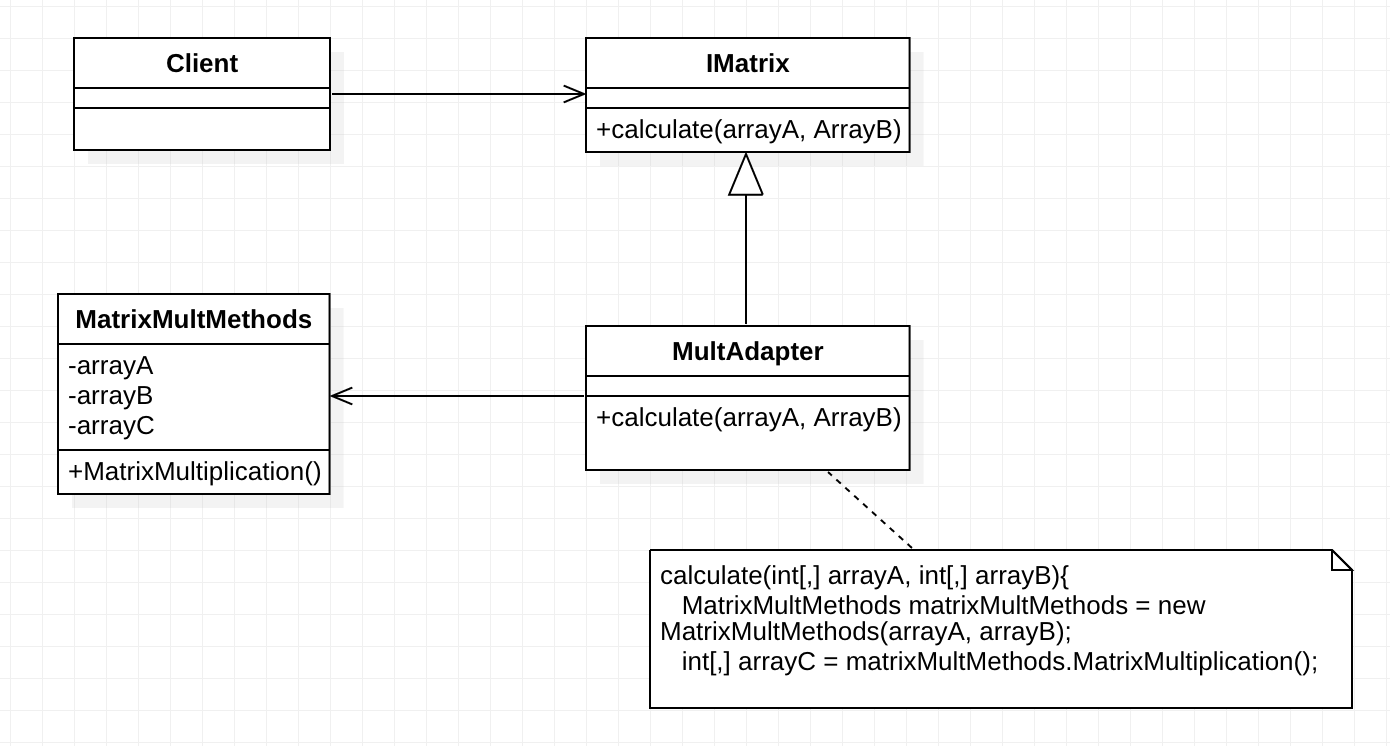


Рисунок 1 – Диаграмма классов с реализацией паттерна «Адаптер»

Программный код на языке C#

using System;

namespace Ref\_5

{

class Program

{

interface IMatrix

{

public void calculate(int[,] arrayA, int[,] arrayB);

}

public class MatrixMultMethods

{

private int[,] matrixA;

private int[,] matrixB;

private int[,] matrixC;

public MatrixMultMethods(int[,] A, int[,] B)

{

matrixA = A;

matrixB = B;

}

private int RowsCount(int[,] matrix)

{

return matrix.GetUpperBound(0) + 1;

}

private int ColumnsCount(int[,] matrix)

{

return matrix.GetUpperBound(1) + 1;

}

public int[,] MatrixMultiplication()

{

if (ColumnsCount(matrixA) != RowsCount(matrixB))

{

throw new Exception("Умножение не возможно! Количество столбцов первой матрицы не равно количеству строк второй матрицы.");

}

Console.WriteLine("Процесс умножения");

matrixC = new int[RowsCount(matrixA), ColumnsCount(matrixB)];

for (var i = 0; i < RowsCount(matrixA); i++)

{

for (var j = 0; j < ColumnsCount(matrixB); j++)

{

matrixC[i, j] = 0;

for (var k = 0; k < ColumnsCount(matrixA); k++)

{

matrixC[i, j] += matrixA[i, k] \* matrixB[k, j];

}

}

}

return matrixC;

}

}

class MultAdapter:IMatrix

{

public void calculate(int[,] arrayA, int[,] arrayB)

{

MatrixMultMethods matrixMultMethods = new MatrixMultMethods(arrayA, arrayB);

int[,] arrayC = matrixMultMethods.MatrixMultiplication();

Console.WriteLine("Matrix multiplication result:");

for (int i = 0; i < arrayC.GetUpperBound(0) + 1; i++)

{

for (int j = 0; j < arrayC.GetUpperBound(1) + 1; j++)

{

Console.Write(arrayC[i, j] + " ");

}

Console.WriteLine();

}

}

}

static void Main(string[] args)

{

int[,] arrayA = { {1,2,3 },{ 2,3,4} };

int[,] arrayB = { { 3, 2 }, { 0, 0 }};

IMatrix matrix = new MultAdapter();

matrix.calculate(arrayB, arrayA);

}

}

}

**Выводы**

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