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 №6

По дисциплине: «Рефакторинг программного обеспечения»

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

**2 Постановка задачи**

3.1. Изучить назначение и структуру паттерна *Цепочка обязанностей* (выполнить в ходе самостоятельной подготовки).

3.2. Применительно к программному продукту, выбранному для рефакторинга, проанализировать возможность использования паттерна

Приведем пример функции диспетчеризации, в которой используются объекты для идентификации запросов. Операция GetKind(), указанная в *Цепочка обязанностей*. Для этого построить диаграмму классов, на диаграмме классов найти класс-клиент, запрос от которого необходимо передавать по цепочке объектов, и классы-получатели запросов, объекты которых целесообразно объединять в цепочку.

3.3. Выполнить перепроектирование системы, использовав паттерн *Цепочка обязанностей*, 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**

Реализована программа, реализующая паттерн «Цепочка обязанностей», диаграмма классов представлена на рисунке 1.

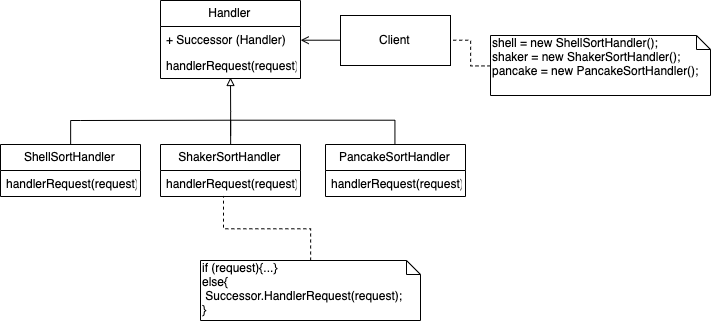


Рисунок 1 – Диаграмма классов с реализацией паттерна «Цепочка обязанностей»

Программный код на языке C#

using System;

namespace ref\_6

{

class Program

{

abstract class Handler

{

public Handler Successor { get; set; }

public abstract void HandlerRequest(int[] request);

}

class ShellSortHandler : Handler

{

public override void HandlerRequest(int[] request)

{

if (request[0] == 1)

{

ShellSort shellSort = new ShellSort();

Console.WriteLine("Shell sort: {0}", string.Join(", ", shellSort.sort(request)));

Console.WriteLine("Chain element: ShellSort");

}

else if (Successor != null)

{

Console.WriteLine("Shell can't sort this array go to the next sorter");

Successor.HandlerRequest(request);

}

}

}

class ShakerSortHandler : Handler

{

public override void HandlerRequest(int[] request)

{

if (request[0] == 2)

{

ShakerSort shakerSort = new ShakerSort();

Console.WriteLine("Shaker sort: {0}", string.Join(", ", shakerSort.sort(request)));

Console.WriteLine("Chain element: ShellSort -> ShakerSort");

}

else if (Successor != null)

{

Console.WriteLine("Shaker can't sort this array go to the next sorter");

Successor.HandlerRequest(request);

}

}

}

class PancakeSortHandler : Handler

{

public override void HandlerRequest(int[] request)

{

if (request[0] == 3)

{

PancakeSort pancakeSort = new PancakeSort();

Console.WriteLine("Pancake sorter: {0}", string.Join(", ", pancakeSort.sort(request)));

Console.WriteLine("Chain element: ShellSort -> ShakerSort -> PancakeSort");

}

else if (Successor != null)

{

Console.WriteLine("Pancake can't sort this array go to the next sorter");

Successor.HandlerRequest(request);

}

}

}

static void Main(string[] args)

{

Handler shell = new ShellSortHandler();

Handler shaker = new ShakerSortHandler();

Handler pancake = new PancakeSortHandler();

shell.Successor = shaker;

shaker.Successor = pancake;

int[] arrayForShell = {1, 77, -9, 6, 3, 8, -9, -1000, 1024};

int[] arrayForShaler = { 2, 7, -9, 8, 3, 45, -90, 88 };

int[] arrayForPancake = { 3, 20, -9, 6, 3, 8, 0, 64 };

Console.WriteLine("Unsorted array: {0}", string.Join(", ", arrayForShell));

shell.HandlerRequest(arrayForShell);

Console.WriteLine();

Console.WriteLine("Unsorted array: {0}", string.Join(", ", arrayForShaler));

shell.HandlerRequest(arrayForShaler);

Console.WriteLine();

Console.WriteLine("Unsorted array: {0}", string.Join(", ", arrayForPancake));

shell.HandlerRequest(arrayForPancake);

}

}

}

**Выводы**

В ходе выполнения лабораторной работы был исследован и применен поведенческий паттерн проектирования – «Цепочка обязанностей». Были получены практические навыки применения поведенческих паттернов при объектно-ориентированном проектировании и рефакторинге ПО.