Лабораторная работа № 4
Разработка программ с использованием структурных паттернов

Цель работы: научиться разрабатывать программы с использованием структурных паттернов проектирования *Адаптер* и *Фасад* из каталога GoF.

1. Теоретические сведения

Шаблон Адаптер

Адаптер – структурный шаблон проектирования, предназначенный для преобразования интерфейса класса к другому интерфейсу, на который рассчитан клиент.

Проблема.

Как обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких классов с разными интерфейсами?

Решение.

Преобразовать исходный интерфейс класса к другому виду с помощью промежуточного объекта-адаптера.

Основными участниками решения являются:

- `Client` – клиент, использующий целевой интерфейс;
- `ITarget` – целевой интерфейс;
- `Adapter` – адаптер, реализующий интерфейс `ITarget`;
- `Adapted` – адаптируемый класс, имеющий интерфейс, несовместимый синтерфейсом клиента.

Работа клиента с адаптируемым объектом происходит следующим образом:

1) клиент обращается с запросом к объекту-адаптеру `Adapter`, вызывая егометод `Request()` через целевой интерфейс `ITarget`;
2) адаптер `Adapter` преобразует запрос в один или несколько вызовов кадаптируемому объекту `Adapted`;
3) клиент получает результаты вызова, не зная ничего о преобразованиях,выполненных адаптером.

Шаблон Адаптер имеет следующие разновидности:

- адаптер объекта применяет для адаптации одного интерфейса к другомукомпозицию объектов адаптируемого класса (рисунок 1).
- адаптер класса использует для адаптации наследование адаптируемомуклассу (рисунок 2).

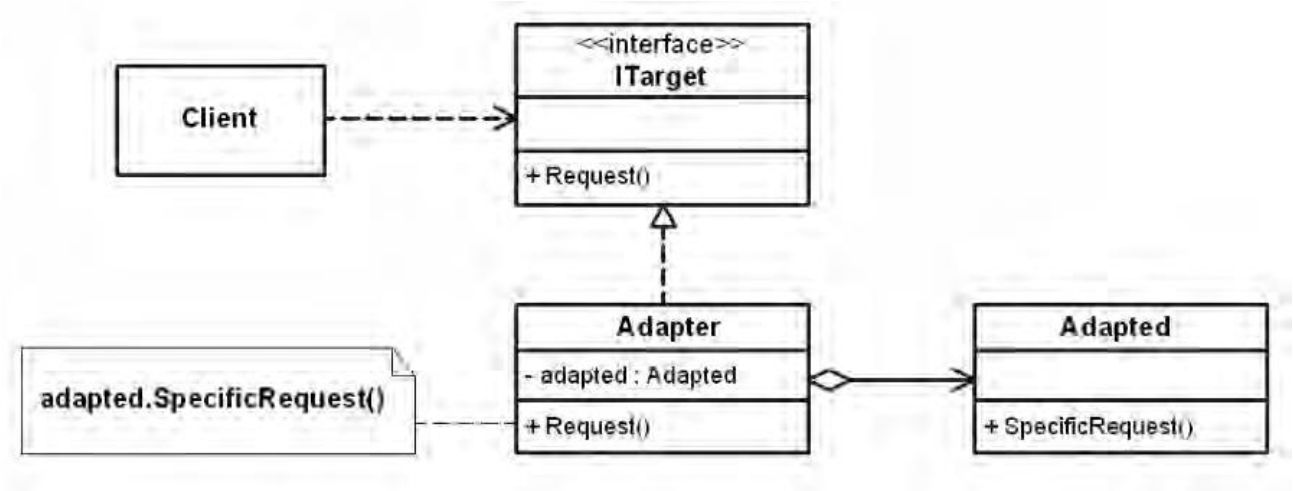


Рисунок 1 – Диаграмма классов шаблона *Адаптер объекта*

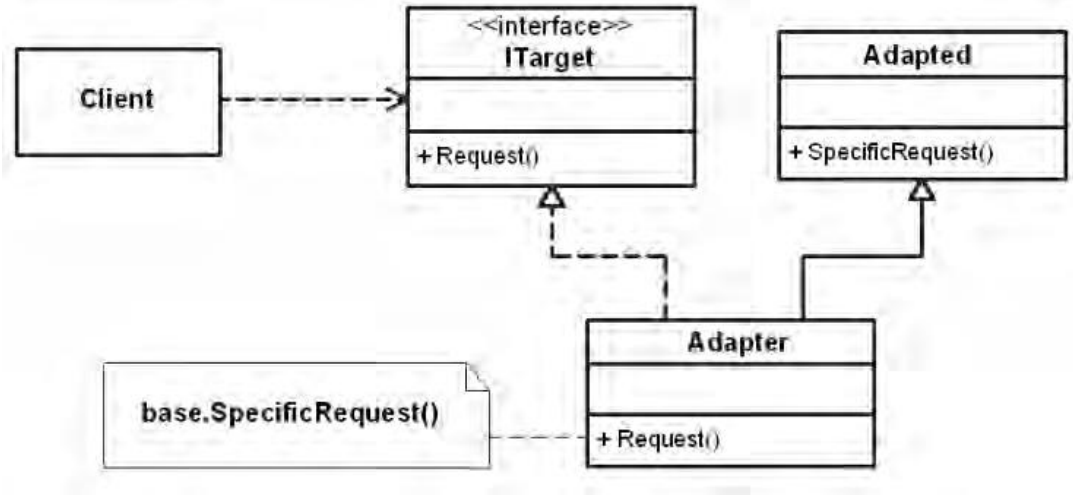


Рисунок 2 – Диаграмма классов шаблона *Адаптер класса*

Результаты.

Система становится независимой от интерфейса внешних классов (компонентов, библиотек). При переходе на использование внешних классов не требуется переделывать всю систему, достаточно переделать один класс Adapter.

Результаты применения адаптеров классов и объектов различаются.

Адаптер объекта:

- позволяет работать объекту `Adapter` со многими адаптируемыми объектами (например, с объектами класса `Adapted` и его производных классов);
- отличается сложностью при замещении операций класса `Adapted` (для этого может потребоваться создать класс производный от `Adapted`, и добавить в класс `Adapter` ссылку на этот производный класс).

Адаптер классов:

- обеспечивает простой доступ к элементам адаптируемого класса, поскольку Adapter является производным классом от `Adapted`;
- характеризуется легкостью изменения адаптером операций адаптируе-

мого класса Adapted;

- обладает возможностью работы только с одним адаптируемым классом (возможность адаптировать классы, производные от Adaptee, отсутствует).

Реализация.

Пример реализации шаблона Адаптер объекта на языке C# представлен ниже.

```
// Представляет целевой интерфейс
public interface ITarget
{
      void Request();
}
// Представляет адаптируемые объекты
class Adapted
{
      public void SpecificRequest()
      {
           Console.WriteLine("Вызван
           SpecificRequest()");
      }
}
// Представляет объекты-адаптеры
public class Adapter : ITarget
{
      Adapted adapted = new
      Adapted();public void
      Request()
      {
           adapted.SpecificRequest();
      }
}
// Клиент
class Client
{
      static void Main(string[] args)
      {
           ITarget target = new
           Adapter();
           target.Request();
      }
}
```

Отличие реализации шаблона *Адаптер* класса будет заключаться только в коде класса *Adapter*.

```
// Представляет объекты-адаптеры
public class Adapter : Adapted, ITarget
{
      public void Request()
```

```
        {
            base.SpecificRequest();
        }
    }
```

Шаблон Фасад

Фасад – структурный шаблон проектирования, позволяющий скрыть сложность подсистемы путем сведения всех возможных вызовов к одному объекту (фасадному объекту), делегирующему их объектам под- системы (рисунок 3).

Проблема.

Как обеспечить унифицированный интерфейс с подсистемой, если нежелательна высокая степень связанности с этой подсистемой или реализация подсистемы может измениться?

Решение.

Определить одну точку взаимодействия с подсистемой – фасадный объект, обеспечивающий единый упрощенный интерфейс с подсистемой и возложить на него обязанность по взаимодействию с классами подсистемы.

Участники решения:

- *Client* – взаимодействует с фасадом и не имеет доступа к классам под- системы;
- *Facade* – перенаправляет запросы клиентов к классам подсистемы;
- *Классы подсистемы* – выполняют работу, порученную объектом *Facade* ничего не зная о существовании фасада, то есть не хранят ссылок на него.

Результаты:

- клиенты изолируются от классов (компонентов) подсистемы, что уменьшает число объектов, с которыми клиенты взаимодействуют, и упрощает работу с подсистемой;
- снижается степень связанности между клиентами и подсистемой, что позволяет изменять классы подсистемы, не затрагивая при этом клиентов.

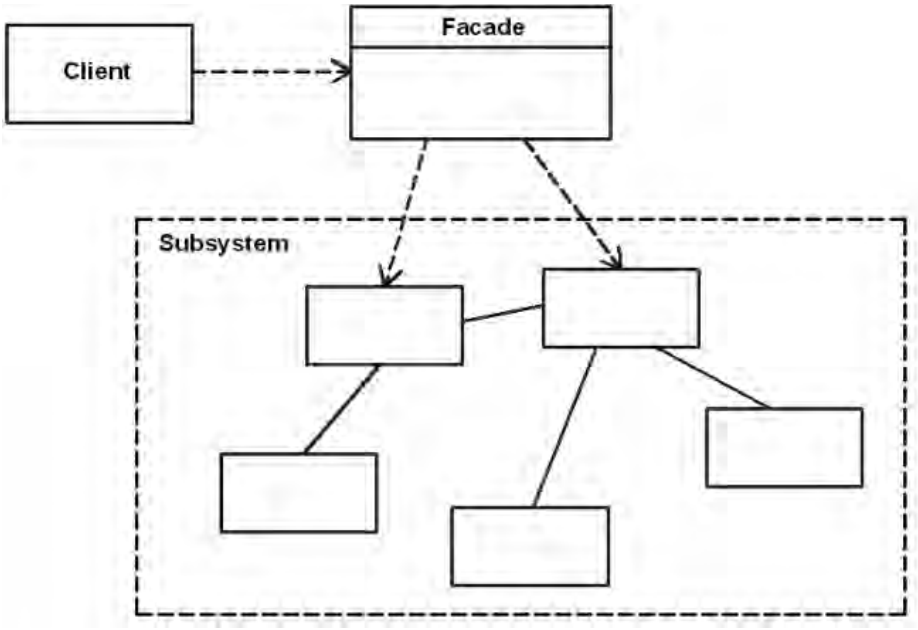


Рисунок 7.3 – Диаграмма классов шаблона Фасад

Реализация.
Пример реализации шаблона Фасад на языке C# представлен ниже.

```
namespace Subsystem // Содержит классы подсистемы
{
      internal class ClassA
      {
           public string A1()
           {
                return "Метод A1() класса ClassA";
           }
           public string A2()
           {
                return "Метод A2() класса ClassA";
           }
}
internal class ClassB
{
      public string B1()
      {
           return "Метод B1() класса ClassB";
      }
      public string B2()
      {
           return "Метод B2() класса ClassB";
      }
}
}
// Представляет фасадные объекты
public class Facade
{
```

```
        Subsystem.C
        lassA    a;
        Subsystem.C
        lassB    b;
        public
        Facade()
        {
             a = new
             Subsystem.ClassA();
             b = new
             Subsystem.ClassB();
        }
        public void F1()
        {
             Console.WriteLine("Метод F1() класса
             Facade:\n"
              +"* {0}\n" +"* {1}\n", a.A1(), b.B2());
        }
        public void F2()
        {
             Console.WriteLine("Метод F2() класса
             Facade:\n"
             +"* {0}\n" +"* {1}\n" + "* {2}\n",
             a.A2(),b.B1(), b.B2());
        }
}
// Клиент
class Client
{
        static void Main(string[] args)
        {
             Facade facade = new
             Facade();facade.F1();
             facade.F2();
        }
}
```

2. Задания к лабораторной работе

Для заданного варианта задания разработать UML-диаграмму классов и диаграмму последовательности.

Разработать консольное приложение (C++, C#). Допускается вводить дополнительные понятия предметной области. В программе предусмотреть тестирование функциональности созданных объектов классов.

Отчет по лабораторной работе – файл формата pdf. Формат имени файла отчета: <НомерГруппы >_<ФамилияСтудента>.pdf. В отчет включить построенные диаграммы и исходный код программы (при необходимости и заголовочные файлы). Формат отчета см. в Приложении. Отчет загрузить в LMS.

При защите лабораторной работы: уметь объяснить логику и детали работы программы; реализацию паттернов проектирования на примере разработанной программы.

1. Изучить пример проектирования программной системы с использованием паттерна Адаптер [Турчин-Архитектура ИС.pdf [Электронный ресурс], с. 117–123].
2. Изучить пример проектирования программной системы с использованием паттерна Фасад [Турчин-Архитектура ИС.pdf [Электронный ресурс], с. 123–133].
3. Разработать библиотеку классов, которая содержит указанные классы (таблица 1), задействованные в паттерне *Адаптер*. Для адаптера объектов атрибуты класса должны быть реализованы как автоматические свойства, а для паттерна *Адаптер классов* – как защищённые поля.

Таблица 1 – Варианты заданий для разработки приложения с использованием паттерна *Адаптер*

Номер варианта	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
1	объектов	`+ CalculateDp(T0 : int, dT : int): double` – определить изменение давления при заданной начальной температуре `T0` и изменении температуры `dT`; `+ ModifMass(dm: double):void` – изменить массу газа в баллоне на величину `dm`; `+ GetData() : string` – возвращает строку с данными об объекте	Баллон с газом. *Атрибуты:* • `Volume : double` – объём баллона, м^3; • `Mass : double` – масса газа, кг; • `Molar : double` – молярная масса газа, кг/моль. *Операции:* `+ GetPressure(T : int) : double` – определить давление в баллоне при заданной температуре газа `T`; `+ AmountOfMatter() : double` – определить количество вещества; `+ ToString() : string` – возвращает строку с данными об объекте
2	классов	`+ ModifVolume(dV : double) : void` – изменить объём баллона на величину `dV`; `+ GetDp(T0 : int, T1 : int) : double` – определить изменение давления при изменении температуры с `T0` до `T1`; `+ Passport() : string` – возвращает строку с данными об объекте	

Уравнение состояния идеального газа (уравнение Менделеева-Клапейрона):

$$pV = \nu RT = \frac{m}{M} RT ;$$

где p – давление газа; V – объём сосуда; ν – количество вещества; m – масса газа; M – молярная масса газа (например, для водорода M = 2,016 г/моль, для азота M = 28 г/моль, для кислорода M = 32 г/моль); T – температура газа; $R \approx 8{,}31$ Дж/(моль·К) – молярная газовая постоянная.

4. Разработать библиотеку классов, которая должна содержать класс-фасад и заданный набор классов (таблица 2). В фасаде необходимо задать ссылки на другие классы библиотеки.
5. Добавить в решение консольное приложение, которое для реализованных паттернов играет роль клиента. Продемонстрировать работу в консольном приложении работу шаблонов проектирования.
6. Для заданных вариантов разработать UML-диаграммы классов и диаграммы последовательности.

Таблица 2 – Варианты заданий для разработки приложения с использованием паттерна *Фасад*

Номер варианта	Данные для разработки приложения на основе шаблона Фасад
1	*Расчёт страхового взноса за недвижимость*. Классы (типы недвижимости): квартира, таун-хаус, коттедж. Параметры: срок страхования, жилплощадь ($м^2$), число проживающих, год постройки здания, износ здания (%)
2	*Расчёт ежедневной нормы потребления килокалорий*. Классы (Тип телосложения): Астеник, Нормостеник, Гиперстеник. Параметры: Рост, Вес, Возраст, Пол Группа физической активности (низкая, средняя и высокая активность)
3	*Расчёт стоимости туристической путевки*. Классы (виды путевок): пляжный отдых, экскурсия, горные лыжи. Параметры: длительность, страна, гостиница (число звезд), рацион питания (двухразовый, трехразовый, всё включено)

Распределение вариантов заданий

№ по списку группы	№ варианта		№ по списку группы	№ варианта	
	Адаптер	Фасад		Адаптер	Фасад
1	1	1	16	2	1
2	2	2	17	1	2
3	1	3	18	2	3
4	2	1	19	1	1
5	1	2	20	2	2
6	2	3	21	1	3
7	1	1	22	2	1
8	2	2	23	1	2
9	1	3	24	2	3
10	2	1	25	1	1
11	1	2	26	2	2
12	2	3	27	1	3
13	1	1	28	2	1
14	2	2	29	1	2
15	1	3	30	2	3

Технологии конструирования программного обеспечения

Отчет по лабораторной работе № 00

Группа: 000-000 **Студент**: Иванов Иван Иванович

Задание на лабораторную работу

<Формулировка задания согласно варианту …… ……..>

Диаграмма классов

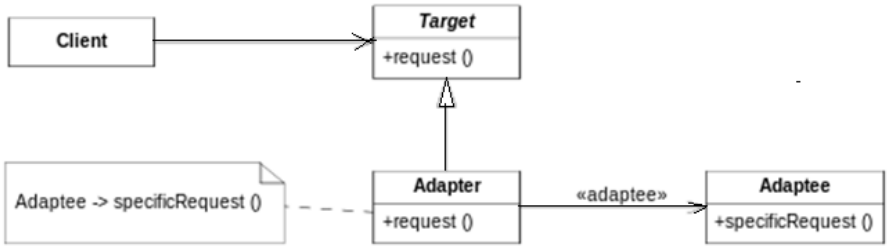
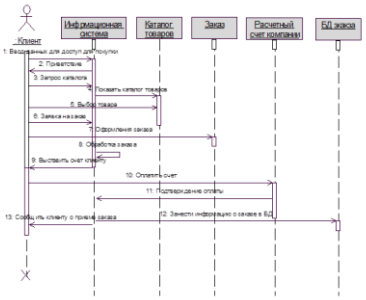


Диаграмма последовательности



Исходный код программы

```
#include "pch.h"
#include "CppUnitTest.h"
#include "../Add/Add.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
	TEST_CLASS(UnitTest1)
	{
	public:

		TEST_METHOD(TestMethod1)
		{
			Assert::AreEqual(2, add(1, 1));
		}
```