Лабораторная работа №7
Изучение и практическое применение структурного шаблона проектирования Адаптер

Адаптер, Adapter или Wrapper/Обёртка — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Задача

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

Решением задачи является преобразование исходного интерфейса 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 с интерфейсом клиента.

Шаблон Адаптер имеет следующие разновидности:

- Адаптер объекта применяет для адаптации одного интерфейса к другому композицию объектов адаптируемого класса.
- Адаптер класса использует для адаптации наследование адаптируемому классу.

Для реализации шаблона Адаптер требуется разработать интерфейс, описывающий целевой программный интерфейс, который умеет использовать клиентский код.

```
/// <summary>
/// Целевой интерфейс, определяющий требуемое поведение
/// </summary>
public interface ITarget
{
    /// <summary>
    /// Метод, который использует клиентский код
    /// </summary>
    void Request();
}
```

Так как этот шаблон используется в случае, если объект, реализующий требуемый функционал, имеет другой интерфейс. Например, имеет метод SpecificRequest() вместо Request():

```
/// <summary>
/// Класс, к которому нужно адаптироваться
/// </summary>
public class Adapted
{
    /// <summary>
    /// Метод, к которому нужно адаптироваться
    /// </summary>
    public void SpecificRequest()
    {
        Console.WriteLine("SpecificRequest");
    }
}
```

Разновидность шаблона Адаптер объекта использует композицию объектов. Т.е. адаптирующий класс хранит ссылку на адаптируемый объект и делегирует вызов методов с изменением параметров при необходимости.

```
/// <summary>
/// Класс, обеспечивающий сопряжение адаптируемого класса
/// с целевым интерфейсом
/// Используется в при использовании адаптера объекта
/// </summary>
public class Adapter : ITarget
{
    /// <summary>
    /// Ссылка на адаптируемый объект
    /// </summary>
    Adapted adapted;

    /// <summary>
    /// Конструктор создания объекта
    /// </summary>
    /// <param name="adapted">Ссылка на адаптируемый объект</param>
    public Adapter(Adapted adapted)
    {
        this.adapted = adapted;
    }

    /// <summary>
    /// Реализация целевого интерфейса.
    /// Делегирует выполнение запроса адаптируемому классу
    /// через ссылку на адаптируемый объект
    /// </summary>
    public void Request()
    {
        adapted.SpecificRequest();
    }
```

```
}
```

Класс адаптер реализует интерфейс ITarget, что гарантирует соответствие программных интерфейсов клиентского кода и адаптера. При вызове метода Request() внутри выполняется вызов специфического метода адаптируемого класса SpecificRequest(). В этом простейшем примере выполняется просто вызов, однако на практике, в случае более сложных классов и интерфейсов, такое действие может сопровождаться дополнением или видоизменением параметров.

Пример клиентского кода:

```
Adapted adapted = new Adapted();

ITarget target = new Adapter(adapted);

target.Request();
```

Обратите внимание, что в клиентском коде экземпляр адаптера сохраняется в переменную, имеющую тип целевого интерфейса. Это не является обязательным условием использования адаптеров, но гарантирует гибкость клиентского кода за счет слабых связей между классами. Так как, создание адаптера может быть вынесено также за клиентский код (например, с использованием шаблона проектирования Фабрика), тогда клиентский код вообще может не знать каким адаптером он пользуется.

```
/// <summary>
/// Класс, обеспечивающий сопряжение адаптируемого класса
/// с целевым интерфейсом посредством множественного наследования
/// Класс наследуется от адаптируемого объекта и целевого интерфейса
/// Используется при использовании адаптера класса
/// </summary>
public class AdapterC : Adapted, ITarget
{
    /// <summary>
    /// Реализация целевого интерфейса.
    /// Делегирует выполнение запроса адаптируемому классу
    /// через вызов метода базового класса (адаптируемого)
    /// </summary>
    public void Request()
    {
        base.SpecificRequest();
    }
}
```

В реализации адаптера класса используется наследование. Адаптер является потомком адаптируемого класса для доступа к реализации

специфического интерфейса, а также реализует целевой интерфейс для его реализации. Такой подход позволяет класс, а не объект. Следовательно, такой подход используется в тех случаях, когда нет экземпляра адаптируемого объекта, а требуется просто адаптировать логику класса.

Клиентский код такого подхода следующий:

```
ITarget target2 = new AdapterC();
target2.Request();
```

Диаграмма классов приведены на рисунках 1.

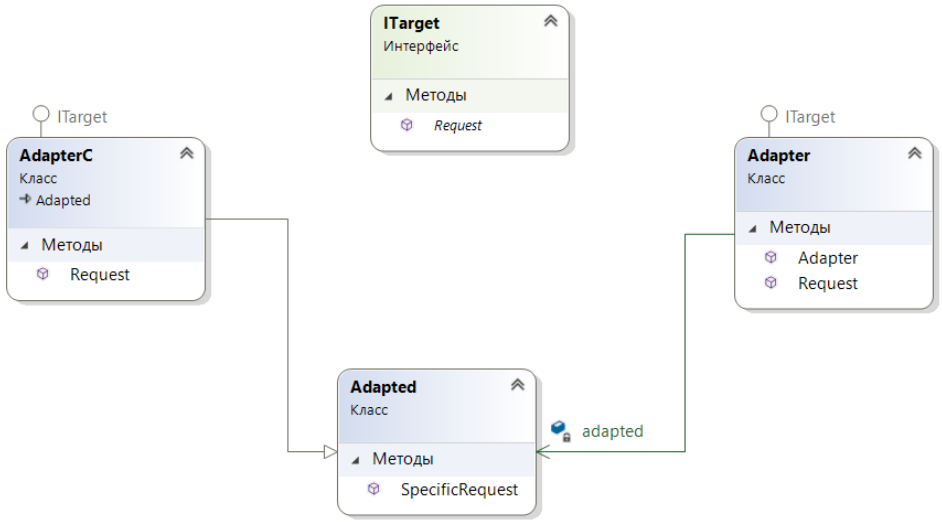

Рисунок 1 – Диаграмма классов шаблона Адаптер

Пример. Требуется разработать приложение на языке C#, в котором для обеспечения совместимости интерфейсов используется шаблон Адаптер. Приложение должно состоять из библиотеки классов, содержащей адаптируемый класс, консольного приложения, выступающего в роли клиента, а также из библиотеки, содержащей целевой интерфейс и класс-адаптер.

Адаптируемый класс: Pool (представляет бассейны с водой).

Атрибуты:

maxH – максимальный уровень воды, см;

curH – текущий уровень воды, см;

poolS – площадь дна бассейна, м2;

holeS – площадь отверстия для слива воды, см2.

Операции:

int CurrentH – возвращает и устанавливает текущий уровень воды в бассейне (свойство);

int GetPourDownT(double h) – определить время, за которое уровень воды опустится до h;

string ToString() – возвращает строку с данными об объекте.

Требуемый интерфейс:

Double CurrentHeight – возвращает текущий уровень воды;

void ModifH(double dH) – изменить уровень воды в бассейне на величину dH, м;

int CalcPourDownT() – определить время, за которое из бассейна выльется вся вода, с;

string GetStringData() – возвращает строку с данными об объекте.

Время вытекания жидкости из сосуда можно найти следующим образом:

$$t = \frac{S}{\sigma}\sqrt{\frac{2}{g}\left(\sqrt{H} - \sqrt{h}\right)}$$

где S – площадь дна сосуда; σ – площадь отверстия в сосуде; H – уровень жидкости относительно дна сосуда; h – уровень, до которого опустится жидкость.

Адаптируемый класс может выглядеть следующим образом:

```
public class Pool
{
    protected int MaxH;
    protected int CurH;
    protected double PoolS;
    protected int HoleS;

    public Pool(int maxH, int curH, double poolS, int holeS)
    {
        MaxH = maxH;
        CurH = curH;
        PoolS = poolS;
        HoleS = holeS;
    }

    public Pool() : this(200, 100, 20, 10)
    {

    }

    public int CurrentH
    {
        get
        {
            return CurH;
        }
```

```
            set
            {
                if ((value >= 0) && (value <= MaxH))
                    CurH = value;
            }
        }

        public int GetPourDown(int h)
        {
            double t = 0;
            double g = 9.8;
            if((h >= 0) && (h < CurH))
            {
                t = (PoolS * 10000 / HoleS) * Math.Sqrt(2 / g) *
(Math.Sqrt(CurH / 100) - Math.Sqrt(h / 100));
            }

            return Convert.ToInt32(t);
        }

        public override string ToString()
        {
            return string.Format("Данные о бассейне: \n" +
                "- макс. уровень воды: {0} см\n" +
                "- текущ. уровень воды: {1} см\n" +
                "- площадь дна бассейна: {2} м2\n" +
                "- площадь отверстия для слива: {3} см2",
                MaxH, CurH, PoolS, HoleS);
        }
    }
```

Целевой интерфейс:

```
    public interface ITarget
    {
        double CurrentH { get; }
        void ModifH(double dH);
        int CalcPourDownT();
        string GetStringData();
    }
```

Класс адаптер:

```
    public class PoolObjAdapter : ITarget
    {
        Pool Pool;

        public PoolObjAdapter(Pool pool)
        {
            Pool = pool;
        }

        public PoolObjAdapter(int maxH, int curH, double poolS, int holeS)
```

```
        {
            Pool = new Pool(maxH, curH, poolS, holeS);
        }

        public double CurrentH => Convert.ToDouble(Pool.CurrentH) / 100;

        public int CalcPourDownT()
        {
            return Pool.GetPourDown(0);
        }

        public string GetStringData()
        {
            return Pool.ToString();
        }

        public void ModifH(double dH)
        {
            Pool.CurrentH += Convert.ToInt32(dH * 100);
        }
    }
```

Диаграмма классов показана на рисунке 2.

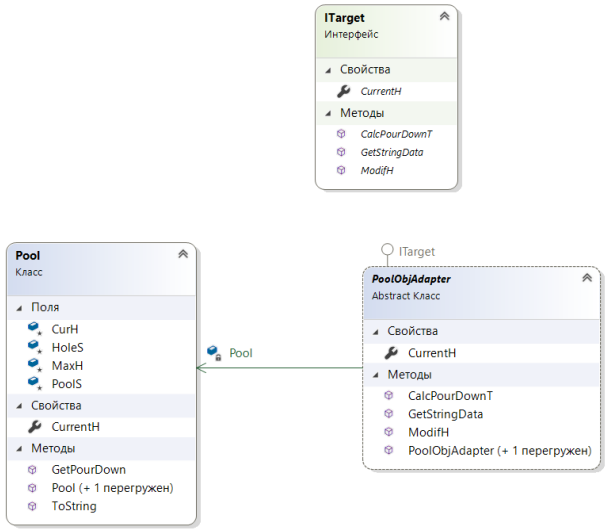

Рисунок 2 – диаграмма класса примера шаблона Адаптер

Как видно, здесь используется адаптер объекта. Все методы и свойства целевого интерфейса просто видоизменяют значения входных параметров и вызывают методы адаптируемого класса.

Клиентский код для проверки работы адаптера может быть следующим:

```
Pool pool = new Pool(1000, 400, 12, 2);

ITarget adapter = new PoolObjAdapter(pool);
```

```
Console.WriteLine(adapter.GetStringData());

Console.WriteLine("Время слива всей воды из бассейна {0} с",
    adapter.CalcPourDownT());

adapter.ModifH(-2.5);

Console.WriteLine("Текущий уровень воды {0} м", adapter.CurrentH);

Console.ReadKey();
```

Задания на лабораторную работу:

1. Реализовать примеры из методических указаний и продемонстрировать их работу.
2. Реализовать иерархию классов с применением шаблона Адаптер объекта и Адаптер класса согласно варианту задания.

№ вар.	Тип адаптера	Требуемый интерфейс	Адаптируемый класс
1,13	Объектов	**Double CalculateDp (int T0, int dT)** – определить изменение давления при заданной начальной температуре T0 и изменении температуры dT; **void ModifMass(double dm)** – изменить массу газа в баллоне на величину dm; **string GetData()** – возвращает строку с данными об объекте	**Баллон с газом.** **double Volume** – объём баллона, м3; **double Mass** – масса газа, кг; **double Molar** – молярная масса газа, кг/моль. **double GetPressure(int T)** – определить давление в баллоне при заданной температуре газа T; **double AmountOfMatter()** – определить количество вещества; **string ToString()** возвращает строку с данными об объекте.
2,14	Классов	**void ModifVolume(double dV)** – изменить объём баллона на величину dV; **double GetDp(int T0, int T)** – Определить изменение давления при изменении температуры с T0 до T1; **string Passport()** – возвращает строку с данными об объекте.	
3,15	Объектов	**void ModifLength(double dl)** – изменить длину подвеса маятника на величину dl; **double GetFreq()** – Определить частоту колебаний; **string Passport()** – возвращает строку с данными об объекте	**Математический маятник** **double a** – амплитуда колебаний; **double d** – длина нерастяжимой нити; **int m** – масса маятника; **double X(int t)** – определить смещение маятника в момент времени t; **double CalculateT()** – определить период колебаний;
4,16	Классов	**void ModifMass(int dm)** – изменить массу маятника на величину dm; **double CalculateW()** – определить циклическую частоту колебаний;	

		string GetData() – возвращает строку с данными об объекте	**string ToString()** – возвращает строку с данными об объекте
5,17	Объектов	**void Translate(double dx, double dy)** – переместить объект на величину dx по оси X и на dy по оси Y; **double GetGravForce(int m, double r, double a)** – определить силу гравитационного взаимодействия с объектом массы m и полярными координатами r и a; **string Passport()** – возвращает строку с данными об объекте	**Астероид** **double m** – масса; **double** x – координата X; **double** y – координата Y. **double Distance(double x, double y)** – определить расстояние от астероида до объекта с координатами x и y; **double Fg(int m, double r)** – определить силу взаимодействия с объектом массы m, расположенного на расстоянии r от данного объекта; **string ToString()** – возвращает строку с данными об объекте
6,18	Классов	**void Move(double dr, double a)** – переместить объект на расстояние dr в направлении, задаваемом углом a; **double CalculateFgrav(int m, double x, double y)** – определить силу гравитационного взаимодействия с объектом массы m и координатами x и y; **string GetData()** – возвращает строку с данными об объекте.	
7,19	Объектов	**double CalculateW(int u)** – определить электрическую энергию конденсатора при известном напряжении u между обкладками; **void ModifS(double dS)** – изменить площадь обкладок конденсатора на величину dS; **string GetData()** – возвращает строку с данными об объекте.	**Плоский конденсатор** **double s** – площадь обкладки конденсатора; **double d** – расстояние между обкладками;

8,20	Классов	**double GetW(int u)** – определить электрическую энергию конденсатора при известном напряжении u между обкладками; **void ModifD(double dD)** – изменить расстояние между обкладками конденсатора на величину dD; **string Passport()** – возвращает строку с данными об объекте.	**double eps** – диэлектрическая проницаемость среды между обкладками. **double Capacity()** – определить ёмкость конденсатора; **double GetCharge(int u)** – определить заряд на обкладках конденсатора при известном напряжении u; **string ToString()** – возвращает строку с данными об объекте.
9,21	Объектов	**double CalculateW()** – определить циклическую частоту колебаний; **void ModifA(double dA)** – изменить амплитуду колебаний на величину dA; **string Passport()** – возвращает строку с данными об объекте.	**Пружинный маятник** double a – амплитуда колебаний; double k – жёсткость пружины; int m – масса маятника. double ElasticForce(int t) – определить силу упругости в момент времени t; double GetFreq() – определить частоту колебаний; string ToString() – возвращает строку с данными об объекте, смещение от положения равновесия
10,22	Классов	**void ModifMass(double dM)** – изменить массу маятника на величину **dM**; **double CalculateT()** – определить период колебаний; **string GetData()** – возвращает строку с данными об объекте.	
11,23	Объектов	**void Move(double dx, double dy)** – переместить объект на величину dx по оси X и на dy по оси Y; **double CalculateForce(double q, double r, double a)** – определить силу электростатического	**Точечный электрический заряд** **double q** – заряд; **double x** – координата X;

		взаимодействия с точечным зарядом q, имеющим полярные координаты r и a; **string GetData()** – возвращает строку с данными об объекте.	**double y** – координата Y. **double F(int m, double r)** – определить силу взаимодействия с объектом массы m, расположенного на расстоянии r от данного объекта; **double R(double x, double y)** – определить расстояние отданного заряда до точки с координатами x и y; **string ToString()** – возвращает строку с данными об объекте.
12,24	Классов	**void Translate(double dr, double a)** – переместить объект на расстояние dr в направлении, задаваемом углом a; **double GetElectrForce(int q, double x, double y)** – определить силу электростатического взаимодействия с точечным зарядом q, расположенном в координатах x и y; **string Passport()** – возвращает строку с данными об объекте.	