Министерство науки высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет «инфокоммуникационных технологий»

Направление подготовки «11.04.02 Инфокоммуникационные технологии и системы связи»



**Отчет**

по практической работе № 4

«Построение и исследование программной системы на основе шаблонов проектирования»

По дисциплине: Проектирование инфокоммуникационных систем

Выполнил:

Студент гр. К4111c

Фёдоров Никита Константинович

Проверил:

Волкодав Владимир Алексеевич

г. Санкт-Петербург

2020 г.

[ВВЕДЕНИЕ 3](#_Toc51596848)

[Упражнение 1. Адаптер (Adapter) 4](#_Toc51596849)

[Упражнение 2. Абстрактная фабрика (Abstract Factory) 11](#_Toc51596850)

[Упражнение 3. Фабричный метод (Factory Method) 19](#_Toc51596851)

[Упражнение 4. Одиночка (Singleton) 26](#_Toc51596852)

[Упражнение 5. Стратегия (Strategy) 29](#_Toc51596853)

[Упражнение 6. Шаблонный метод (Template Method) 35](#_Toc51596854)

[Упражнение 7. Фасад (Facade) 41](#_Toc51596855)

[Упражнение 8. Цепочка обязанностей (Chain of Responsibility) 48](#_Toc51596856)

[Упражнение 9. Команда (Command) 51](#_Toc51596857)

[Упражнение 10. Декоратор (Decorator) 58](#_Toc51596858)

[Литература 65](#_Toc51596859)

# ВВЕДЕНИЕ

Цель практической работы:

* Изучить шаблоны проектирования GoF (Gang-of-Four) и закрепить основы разработки систем на их основе;
* Освоить IDE MS Visual Studio в части разработки диаграммы классов.

Задачи на выполнение практической работы:

* Изучить шаблоны проектирования GoF, определить особенности их применения в программных системах.
* Выполнить упражнения и контрольные задания руководства.

# Упражнение 1. Адаптер (Adapter)

**Adapter (Адаптер)** – структурный паттерн. Он обеспечивает возможность совместной работы классов с несовместимыми интерфейсами, позволяя разработчикам реализовывать наборы полиморфных классов, обеспечивающих альтернативные реализации существующего класса.

В этом упражнении реализуется шаблон Адаптер для следующей модели. Есть игрок, который играет в «кости» (бросает кубик). Но в какой-то момент ему приходится решать: бросать в очередной раз кубик или прекратить игру. В этом случае он хочет выбрать решение, кинув монету, но в классе игрока использование класса монеты не предусмотрено, поэтому необходимо создать адаптер.

В этом упражнении применяется адаптер объекта как вариант работы с адаптируемым объектом, при котором используется композиция или агрегация, т.е. адаптер содержит экземпляр адаптируемого объекта или его ссылку. Конкретно в упражнении применяется агрегация.

Классы проекта:

* Program;
* Gamer;
* Kost;
* Monet;
* AdapterGame.

Интерфейсы проекта:

* IGame.

**Интерфейс IGame:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Adapter

{

interface IGame

{

int Brosok();

}

}

**Класс Program:**

using System;

namespace Adapter

{

class Program

{

static void Main(string[] args)

{

Kost kubik = new Kost();

Gamer g1 = new Gamer("Никита");

Console.WriteLine("Выпало очков {0} для игрока {1}", g1.SeansGame(kubik), g1.ToString());

Monet mon = new Monet();

IGame bmon = new AdapterGame(mon);

Console.WriteLine("Монета показала \"{0}\" для игрока {1}", g1.SeansGame(bmon), g1.ToString());

}

}

}

**Класс Gamer:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Adapter

{

class Gamer

{

public string Name { get; set; }

public Gamer(string name)

{

Name = name;

}

public override string ToString()

{

return Name;

}

public int SeansGame(IGame ig)

{

return ig.Brosok();

}

}

}

**Класс AdapterGame:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Adapter

{

class AdapterGame : IGame

{

Monet mot;

public AdapterGame(Monet mt)

{

mot = mt;

}

public int Brosok()

{

return mot.BrosokM();

}

}

}

**Класс Monet:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Adapter

{

class Monet

{

Random r;

public Monet()

{

r = new Random();

}

public int BrosokM()

{

//Случаное число 1 или 2.

int res = r.Next(2) + 1;

return res;

}

}

}

**Класс Kost:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Adapter

{

class Kost : IGame

{

Random r;

public Kost()

{

r = new Random();

}

public int Brosok()

{

// Случайное число от 1 до 6.

int res = r.Next(6) + 1;

return res;

}

}

}

**Результат работы программы (рисунок 1):**

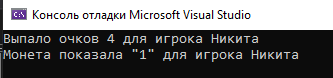


Рисунок 1 – Результат работы программы

**Контрольное задание:**

Разрабатывается система климат-контроля, предназначенная для автоматического поддержания температуры окружающего пространства в заданных пределах.

Важным компонентом такой системы является температурный датчик, с помощью которого измеряют температуру окружающей среды для последующего анализа.

Для этого датчика уже имеется готовое программное обеспечение от сторонних разработчиков, представляющее собой некоторый класс с соответствующим интерфейсом.

Однако использовать этот класс непосредственно не удастся, так как показания датчика снимаются в градусах Фаренгейта.

Требуется разработать адаптер, преобразующий температуру в шкалу Цельсия.

Классы проекта:

* Program;
* MeasureFahrenheit;
* MeasureCelsius;
* Measure;
* AdapterGrad.

Интерфейсы проекта:

* IGrad.

**Интерфейс IGrad:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Ex1Contr

{

interface IGrad

{

int Grad();

}

}

**Класс MeasureFahrenheit:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Ex1Contr

{

class MeasureFahrenheit : IGrad

{

Random r;

public MeasureFahrenheit()

{

r = new Random();

}

public int Grad()

{

// Случайная температура.

int res = r.Next(-100, 100);

return res;

}

}

}

**Класс MeasureCelsius:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Ex1Contr

{

class MeasureCelsius

{

Random r;

public MeasureCelsius()

{

r = new Random();

}

public int GradC()

{

// Случайная температура.

int res = r.Next(-30, 30);

return res;

}

}

}

**Класс Measure:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Ex1Contr

{

class Measure

{

public string Name { get; set; }

public Measure(string name)

{

Name = name;

}

public override string ToString()

{

return Name;

}

public int MeasureTemperature(IGrad ig)

{

return ig.Grad();

}

}

}

**Класс AdapterGrad:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Ex1Contr

{

class AdapterGrad : IGrad

{

MeasureCelsius cel;

public AdapterGrad(MeasureCelsius c)

{

cel = c;

}

public int Grad()

{

return cel.GradC();

}

}

}

**Класс Program:**

using System;

namespace Ex1Contr

{

class Program

{

static void Main(string[] args)

{

MeasureFahrenheit fahr = new MeasureFahrenheit();

Measure m1 = new Measure("Fahrenheit: ");

Console.WriteLine("{0} {1} deg", m1.ToString(), m1.MeasureTemperature(fahr));

MeasureCelsius cel = new MeasureCelsius();

Measure m2 = new Measure("Celsius: ");

IGrad measCel = new AdapterGrad(cel);

Console.WriteLine("{0} {1} deg", m2.ToString(), m2.MeasureTemperature(measCel));

}

}

}

**Результат выполнения программы (рисунок 2):**

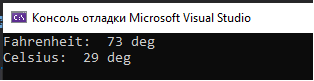


Рисунок 2 – Результат выполнения программы

**Вывод:**

Шаблон проектирования Adapter обеспечивает возможность совместной работы классов с несовместимыми интерфейсами, позволяя разработчикам реализовывать наборы полиморфных классов, обеспечивающих альтернативные реализации существующего класса. Если есть класс, обладающий требуемыми методами и свойствами, то при необходимости всегда можно воспользоваться шаблоном Adapter для приведения его интерфейса к нужному виду.

# Упражнение 2. Абстрактная фабрика (Abstract Factory)

**Паттерн Абстрактная фабрика (Abstract Factory)** – порождающий шаблон проектирования, предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов, что позволяет разработчику создать интерфейс для объектов, каким-либо образом связанных между собой, причем не требуется указывать конкретные классы, поскольку работать с каждым из них можно будет через этот интерфейс.

Таким образом, с помощью такой фабрики удастся создавать группы объектов, реализующих общее поведение.

В этом упражнении реализуется шаблон Абстрактная фабрика для модели фабрики производства автомобилей.

Классы проекта:

* Program;
* CarFactory;
* AbstractCar;
* AbstractEngine;
* FordFactory;
* FordCar;
* FordEngine;
* Client.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AbstractFactory

{

class Program

{

static void Main(string[] args)

{

CarFactory ford\_car = new FordFactory();

Client c1 = new Client(ford\_car);

Console.WriteLine("Max speed {0} is {1} km/h", c1.ToString(), c1.RunMaxSpeed());

}

}

}

**Класс CarFactory:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

abstract class CarFactory

{

public abstract AbstractCar CreateCar();

public abstract AbstractEngine CreateEngine();

}

}

**Класс AbstractCar:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

abstract class AbstractCar

{

public string Name { get; set; }

public abstract int MaxSpeed(AbstractEngine engine);

}

}

**Класс AbstractEngine:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

abstract class AbstractEngine

{

public int max\_speed { get; set; }

}

}

**Класс FordFactory:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class FordFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new FordCar("Ford");

}

public override AbstractEngine CreateEngine()

{

return new FordEngine();

}

}

}

**Класс FordCar:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class FordCar : AbstractCar

{

public FordCar(string name)

{

Name = name;

}

public override int MaxSpeed(AbstractEngine engine)

{

int ms = engine.max\_speed;

return ms;

}

public override string ToString()

{

return "Car " + Name;

}

}

}

**Класс FordEngine:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class FordEngine : AbstractEngine

{

public FordEngine()

{

max\_speed = 220;

}

}

}

**Класс Client:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class Client

{

private AbstractCar abstractCar;

private AbstractEngine abstractEngine;

public Client(CarFactory car\_factory)

{

abstractCar = car\_factory.CreateCar();

abstractEngine = car\_factory.CreateEngine();

}

public int RunMaxSpeed()

{

return abstractCar.MaxSpeed(abstractEngine);

}

public override string ToString()

{

return abstractCar.ToString();

}

}

}

**Результат выполнения программы (рисунок 3):**

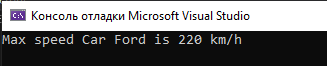


Рисунок 3 – Результат выполнения программы

**Контрольное задание:**

В разработанное приложение необходимо добавить класс для новой конкретной фабрики, создающей новый автомобиль, например, Audi.

Добавить в конфигурацию автомобиля новое свойство – тип кузова (CarBody).

Добавленные классы проекта:

* AbstractCarBody;
* FordCarBody;
* AudiFactory;
* AudiCar;
* AudiCarBody;
* AudiEngine;
* Program.

**Класс AbstractCarBody:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

abstract class AbstractCarBody

{

public string carbody { get; set; }

}

}

**Класс AudiFactory:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class AudiFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new AudiCar("Audi");

}

public override AbstractEngine CreateEngine()

{

return new AudiEngine();

}

public override AbstractCarBody CreateCarBody()

{

return new AudiCarBody();

}

}

}

**Класс AudiCar:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class AudiCar : AbstractCar

{

public AudiCar(string name)

{

Name = name;

}

public override int MaxSpeed(AbstractEngine engine)

{

int ms = engine.max\_speed;

return ms;

}

public override string CarBody(AbstractCarBody carbody)

{

string cb = carbody.carbody;

return cb;

}

public override string ToString()

{

return "Car " + Name;

}

}

}

**Класс AudiCarBody:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class AudiCarBody : AbstractCarBody

{

public AudiCarBody()

{

carbody = "Cabriolet";

}

}

}

**Класс AudiEngine:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace AbstractFactory

{

class AudiEngine : AbstractEngine

{

public AudiEngine()

{

max\_speed = 300;

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AbstractFactory

{

class Program

{

static void Main(string[] args)

{

CarFactory ford\_car = new FordFactory();

Client c1 = new Client(ford\_car);

Console.WriteLine("Car body: {0} Max speed {1} is {2} km/h", c1.CarBody(), c1.ToString(), c1.RunMaxSpeed());

CarFactory audi\_car = new AudiFactory();

Client c2 = new Client(audi\_car);

Console.WriteLine("Car body: {0} Max speed {1} is {2} km/h", c2.CarBody(), c2.ToString(), c2.RunMaxSpeed());

}

}

}

**Результат выполнения программы (рисунок 4):**

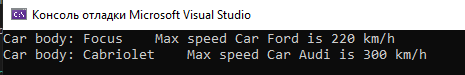


Рисунок 4 – Результат выполнения программы

**Вывод:**

Паттерн Abstract Factory применяется как интерфейс для создания взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов, что позволяет создать интерфейс для связанных между собой объектов без указания конкретных классов, поскольку работать с каждым из них можно будет через этот интерфейс.

# Упражнение 3. Фабричный метод (Factory Method)

**Фабричный метод (Factory Method)** – порождающий шаблон проектирования, предоставляющий подклассам абстрактный интерфейс (набор методов) для создания экземпляров некоторого класса (объекта-продукта). В момент создания наследники могут самостоятельно принять решение о том, экземпляр какого конкретного класса-продукта создать.

Иными словами, данный шаблон делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Таким образом, паттерн Factory Method позволяет базовым абстрактным классам передать ответственность за создание объектов-продуктов своим производным классам.

В этом упражнении реализуется паттерн «Фабричный метод» для транспортной компании, предоставляющей различные услуги клиентам, например, такси и мелкогабаритные грузовые перевозки.

Классы проекта:

* Program;
* TransportService;
* TransportCompany;
* TaxiService;
* Shipping;
* TaxiTransCom;
* ShipTransCom.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FactoryMethod

{

class Program

{

static void Main(string[] args)

{

TransportCompany trCom = new TaxiTransCom("Служба такси");

TransportService compService = trCom.Create("Такси", 1);

double dist = 15.5;

Print(compService, dist);

TransportCompany gCom = new ShipTransCom("Служба перевозок");

compService = gCom.Create("Грузоперевозки", 2);

double distg = 150.5;

Print(compService, distg);

}

private static void Print(TransportService compTax, double distg)

{

Console.WriteLine("Компания {0}, расстояние {1}, стоимость: {2}",

compTax.ToString(), distg, compTax.CostTransportation(distg));

}

}

}

**Класс TransportService:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

abstract class TransportService

{

public string Name { get; set; }

public TransportService(string name)

{

Name = name;

}

abstract public double CostTransportation(double distance);

}

}

**Класс TransportCompany:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

abstract class TransportCompany

{

public string Name { get; set; }

public TransportCompany(string n)

{

Name = n;

}

public override string ToString()

{

return Name;

}

// фабричный метод

abstract public TransportService Create(string n, int k);

}

}

**Класс TaxiService:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class TaxiServices : TransportService

{

public int Category { get; set; }

public TaxiServices(string name, int cat) : base(name)

{

Category = cat;

}

public override double CostTransportation(double distance)

{

return distance \* Category;

}

public override string ToString()

{

string s = String.Format("Фирма {0}, поездка категории {1}", Name, Category);

return s;

}

}

}

**Класс Shipping:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class Shipping : TransportService

{

public double Tariff { get; set; }

public Shipping(string name, int taff) : base(name)

{

Tariff = taff;

}

public override double CostTransportation(double distance)

{

return distance \* Tariff;

}

public override string ToString()

{

string s = String.Format("Фирма {0}, доставка по тарифу {1}", Name, Tariff);

return s;

}

}

}

**Класс TaxiTransCom:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class TaxiTransCom : TransportCompany

{

public TaxiTransCom(string name) : base(name)

{ }

public override TransportService Create(string n, int c)

{

return new TaxiServices(Name, c);

}

}

}

**Класс ShipTransCom:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class ShipTransCom : TransportCompany

{

public ShipTransCom(string name) : base(name)

{ }

public override TransportService Create(string n, int t)

{

return new Shipping(Name, t);

}

}

}

**Результат выполнения программы (рисунок 5):**

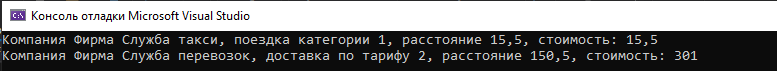


Рисунок 5 – Результат выполнения программы

**Контрольное задание:**

В разработанное приложение необходимо добавить поддержку новой услуги, например, «пьяный водитель».

Классы проекта:

* AlcDriver;
* AlcDriverCom;
* Program.

**Класс AlcDriver:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class AlcDriver : TransportService

{

public double Tariff { get; set; }

public AlcDriver(string name, int taff) : base(name)

{

Tariff = taff;

}

public override double CostTransportation(double distance)

{

return distance \* Tariff;

}

public override string ToString()

{

string s = String.Format("Фирма {0}, поездка по тарифу {1}", Name, Tariff);

return s;

}

}

}

**Класс AlcDriverCom:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace FactoryMethod

{

class AlcDriverCom : TransportCompany

{

public AlcDriverCom(string name) : base(name)

{ }

public override TransportService Create(string n, int t)

{

return new AlcDriver(Name, t);

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FactoryMethod

{

class Program

{

static void Main(string[] args)

{

TransportCompany trCom = new TaxiTransCom("Служба такси");

TransportService compService = trCom.Create("Такси", 1);

double dist = 15.5;

Print(compService, dist);

TransportCompany gCom = new ShipTransCom("Служба перевозок");

compService = gCom.Create("Грузоперевозки", 2);

double distg = 150.5;

Print(compService, distg);

TransportCompany alcCom = new AlcDriverCom("Пьяный водитель");

compService = alcCom.Create("Пьяный водитель", 2);

double distalc = 300.5;

Print(compService, distalc);

}

private static void Print(TransportService compTax, double distg)

{

Console.WriteLine("Компания {0}, расстояние {1}, стоимость: {2}",

compTax.ToString(), distg, compTax.CostTransportation(distg));

}

}

}

**Результат выполнения программы (рисунок 6):**

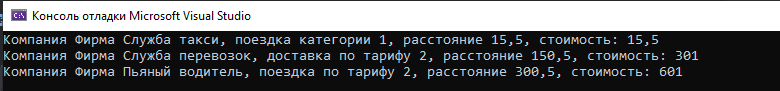


Рисунок 6 – Результат выполнения программы

**Вывод:**

Шаблон проектирования Factory Method (фабричный метод) – удобный способ проектирования базовых абстрактных классов, передающих ответственность за создание наследников своим родительским классам.

# Упражнение 4. Одиночка (Singleton)

Одиночка (Singleton, Синглтон) – порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа. Сам класс контролирует то, что у него есть только один экземпляр, может запретить создание дополнительных экземпляров, перехватывая запросы на создание новых объектов и он же способен предоставить доступ к своему экземпляру.

В этом упражнении создается приложение, в котором ведется протоколирование данных в специальный файл – журнал. В первой версии приложения при необходимости записи данных каждый раз будет создаваться объект класса журнала. Затем по шаблону «Одиночка» изменяется класс журнала, тем самым гарантируется, что у этого класса будет создаваться единственный экземпляр и что этот экземпляр будет легко доступен в любой точке приложения.

Классы проекта:

* Log;
* Operation;
* Program.

**Класс Log:**

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace Singleton

{

class Log

{

Log() { }

static Lazy<Log> myLog = new Lazy<Log>(() => new Log());

public static Log MyLog

{

get

{

return myLog.Value;

}

}

public void LogExecution(string mes)

{

using (StreamWriter w = File.AppendText("log.txt"))

{

Loger(mes, w);

w.Close();

}

}

private static void Loger(string logMessage, TextWriter w)

{

w.Write("\r\nLog Entry : ");

w.WriteLine("{0} {1}", DateTime.Now.ToLongTimeString(), DateTime.Now.ToLongDateString());

w.WriteLine("Действие: {0}", logMessage);

w.WriteLine("-------------------------------");

}

}

}

**Класс Operation:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Singleton

{

class Operation

{

public static double Run(char operationCode, int operand)

{

Log lg2 = Log.MyLog;

double rez = 0;

switch (operationCode)

{

case '+':

rez += operand;

lg2.LogExecution("Сложение " + operand);

break;

case '-':

rez -= operand;

lg2.LogExecution("Вычитание " + operand);

break;

case '\*':

rez \*= operand;

break;

case '/':

case ':':

rez /= operand;

break;

}

return rez;

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Singleton

{

class Program

{

static void Main(string[] args)

{

Log lg = Log.MyLog;

lg.LogExecution("Метод Main()");

double op = Operation.Run('-', 35);

op = Operation.Run('+', 30);

}

}

}

**Результат выполнения программы (рисунок 7):**



Рисунок 7 – Результат выполнения программы

**Вывод:**

Данный шаблон проектирования Одиночка (Singleton, Синглтон) гарантирует, что для определенного класса будет создан только один объект, и предоставляющий глобальную точку доступа к этому экземпляру. Такой паттерн можно использовать для удобства создания документов.

# Упражнение 5. Стратегия (Strategy)

Правила вычислений каких-либо свойств (показателей) может постоянно изменяться в силу самых различных причин (политик). В связи с этим возникает вопрос: как спроектировать изменяемые, но надежные алгоритмы или правила? Одним из возможных решений является определения для каждого алгоритма (правила) отдельный класс со стандартным интерфейсом в рамках паттерна «Стратегия».

Стратегия – это поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс. После чего алгоритмы можно взаимозаменять прямо во время исполнения программы.

Для применение данного паттерна необходимо выполнить следующие действия:

1. Определить алгоритм, который подвержен частым изменениям или алгоритм, имеющий несколько вариаций, которые выбираются во время выполнения программы.
2. Создать интерфейс стратегий, описывающий этот алгоритм. Он должен быть общим для всех вариантов алгоритма.
3. Указать вариации алгоритма в собственных классах, реализующие этот интерфейс.
4. В классе контекста создать поле для хранения ссылки на текущий объект-стратегию, а также метод для её изменения. Контекст должен работать с этим объектом только через общий интерфейс стратегий.
5. При применении паттерна клиент контекста должен передать в контекст, соответствующий требуемому поведению объект-стратегию.

В этом упражнении создается приложение, в котором реализуется на основе паттерна «Стратегия» работа с различными видами сортировки. В целом стратегия включает в себя выбор сортировки определенного типа (вставками, пузырьковая, выбором) в зависимости от сортируемых данных с целью эффективного использования аппаратных ресурсов.

Классы проекта:

* Program;
* StrategySort;
* InsertionSort;
* SelectionSort;
* Context.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class Program

{

static void Main(string[] args)

{

int[] arr1 = { 31, 15, 10, 2, 4, 2, 14, 23, 12, 66 };

StrategySort sort = new SelectionSort();

Context context = new Context(sort, arr1);

context.Sort();

context.PrintArray();

int[] arr2 = { 1, 5, 10, 2, 4, 12, 14, 23, 12, 66 };

sort = new InsertionSort();

context = new Context(sort, arr2);

context.Sort();

context.PrintArray();

}

}

}

**Класс StrategySort:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

abstract class StrategySort

{

public string Title { get; set; }

public abstract void Sort(int[] array);

}

}

**Класс InsertionSort:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class InsertionSort : StrategySort

{

public InsertionSort()

{

Title = "Сортировка вставками";

}

public override string ToString()

{

return Title;

}

public override void Sort(int[] array)

{

for (int i = 1; i < array.Length; i++)

{

int j = 0;

int buffer = array[i];

for (j = i - 1; j >= 0; j--)

{

if (array[j] < buffer)

break;

array[j + 1] = array[j];

}

array[j + 1] = buffer;

}

}

}

}

**Класс SelectionSort:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class SelectionSort : StrategySort

{

public SelectionSort()

{

Title = "Сортировка выбором";

}

public override string ToString()

{

return Title;

}

public override void Sort(int[] array)

{

for (int i = 0; i < array.Length - 1; i++)

{

int k = i;

for (int j = i + 1; j < array.Length; j++)

if (array[k] > array[j])

k = j;

if (k != i)

{

int temp = array[k];

array[k] = array[i];

array[i] = temp;

}

}

}

}

}

**Класс Context:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class Context

{

StrategySort strategy;

int[] array;

public Context(StrategySort strategy, int[] array)

{

this.strategy = strategy;

this.array = array;

}

public void Sort()

{

strategy.Sort(array);

}

public void PrintArray()

{

Console.WriteLine(strategy.ToString());

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

Console.Write(array[i] + " ");

Console.WriteLine();

}

}

}

**Результат выполнения программы (рисунок 8):**

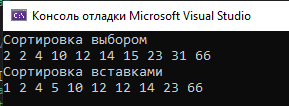


Рисунок 8 – Результат выполнения программы

**Контрольное задание:**

Добавленный класс:

* BubbleSort;
* Program.

**Класс BubbleSort:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class BubbleSort : StrategySort

{

public BubbleSort()

{

Title = "Сортировка пузырьком";

}

public override string ToString()

{

return Title;

}

public override void Sort(int[] array)

{

int temp;

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

{

for (int j = i + 1; j < array.Length; j++)

{

if (array[i] > array[j])

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

}

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Strategy

{

class Program

{

static void Main(string[] args)

{

int[] arr1 = { 31, 15, 10, 2, 4, 2, 14, 23, 12, 66 };

StrategySort sort = new SelectionSort();

Context context = new Context(sort, arr1);

context.Sort();

context.PrintArray();

int[] arr2 = { 1, 5, 10, 2, 4, 12, 14, 23, 12, 66 };

sort = new InsertionSort();

context = new Context(sort, arr2);

context.Sort();

context.PrintArray();

int[] arr3 = { 1, 8, 4, 7, 3, 7, 1, 6 };

sort = new BubbleSort();

context = new Context(sort, arr3);

context.Sort();

context.PrintArray();

}

}

}

**Результат выполнения программы (рисунок 9):**

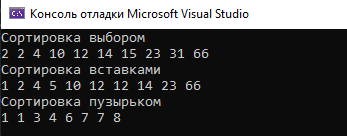


Рисунок 9 – Результат выполнения программы

**Вывод:**

Паттерн Стратегия — это поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс. Этот шаблон проектирования может применяться, когда в одном и том же месте в зависимости от текущего состояния системы (или её окружения) должны использоваться различные алгоритмы.

# Упражнение 6. Шаблонный метод (Template Method)

Шаблонный метод – это поведенческий паттерн проектирования, который предлагает разбить алгоритм на последовательность шагов, описать шаги в отдельных методах и вызывать их в одном «шаблонном» методе друг за другом.

Для применение данного паттерна необходимо выполнить следующие действия:

1. Выполнить декомпозицию алгоритма. Определить, какие шаги будут стандартными для всех вариаций алгоритма, а какие – изменчивыми.
2. Создать абстрактный базовый класс. Определить в нём шаблонный метод. Этот метод должен состоять из вызовов шагов алгоритма.
3. Добавить в абстрактный класс методы для каждого из шагов алгоритма. Стандартные шаги должны иметь реализацию по умолчанию. Изменяемые шаги должны быть объявлены абстрактными. Их нужно будет реализовать в подклассах.
4. Создать конкретные классы, унаследовав их от абстрактного класса. Реализовать в них все недостающие шаги.

В этом упражнении создается приложение, в котором реализуется на основе паттерна «Шаблонный метод» алгоритм работы с различными видами прогрессии. Алгоритм включает в себя настройку параметров прогрессии, генерирование прогрессии по соответствующему правилу, обработка данных (расчет среднего значения, наибольшего, наименьшего и т.д.) и вывод результатов на экран.

Классы проекта:

* Program;
* Progression;
* ArithmeticProgression;
* GeometricProgression.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace TemplateMethod

{

class Program

{

static void Main(string[] args)

{

Progression val = new ArithmeticProgression(1, 30, 5);

val.TemplateMethod();

Progression geom = new GeometricProgression(1, 300, 4);

geom.TemplateMethod();

}

}

}

**Класс Progression:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace TemplateMethod

{

abstract class Progression

{

public int First { get; set; }

public int Last { get; set; }

public int H { get; set; }

public List<int> progList;

public Progression(int first, int last, int h)

{

First = first;

Last = last;

H = h;

progList = new List<int>();

}

public void TemplateMethod()

{

InitializeProgression(First, Last, H);

Progress();

Print(progList);

}

private void Print(List<int> progList)

{

Console.WriteLine("Последовательность:");

foreach (var item in progList)

{

Console.Write(" " + item);

}

Console.WriteLine();

}

private void InitializeProgression(int a, int b, int h)

{

First = a;

Last = b;

H = h;

}

public abstract void Progress();

}

}

**Класс ArithmeticProgression:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace TemplateMethod

{

class ArithmeticProgression : Progression

{

public ArithmeticProgression(int f, int l, int h) : base(f, l, h) { }

public override void Progress()

{

int fF = First;

do

{

progList.Add(fF);

fF = fF + H;

}

while (fF < Last);

}

}

}

**Класс GeometricProgression:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace TemplateMethod

{

class GeometricProgression : Progression

{

public GeometricProgression(int f, int l, int h) : base(f, l, h) { }

public override void Progress()

{

int fe = First;

do

{

progList.Add(fe);

fe = fe \* H;

}

while (fe < Last);

}

}

}

**Результат выполнения программы (рисунок 10):**

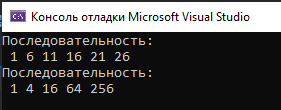


Рисунок 10 – Результат выполнения программы

**Контрольное задание:**

Использован паттерн «Шаблонный метод» для проектирования сложного алгоритма: процесс стрижки. Создан абстрактный базовый класс и определен в нём шаблонный метод, состоящий из вызовов шагов алгоритма.

Классы проекта:

* Progression;
* Client;
* Program.

**Класс Progression:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Barbershop

{

abstract class Progression

{

public bool HairCut { get; set; }

public bool HairColor { get; set; }

public bool HairWashing { get; set; }

public int time;

public Progression(bool hc, bool hcl, bool hw)

{

HairCut = hc;

HairColor = hcl;

HairWashing = hw;

}

public void TemplateMethod()

{

InitProgression(HairCut, HairColor, HairWashing);

Progress();

Print(time);

}

private void Print(int time)

{

Console.WriteLine("Время сеанса {0} мин", time);

Console.WriteLine();

}

private void InitProgression(bool a, bool b, bool c)

{

HairCut = a;

HairColor = b;

HairWashing = c;

}

public abstract void Progress();

}

}

**Класс Client:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Barbershop

{

class Client : Progression

{

public Client(bool a, bool b, bool c) : base(a, b, c) { }

public override void Progress()

{

bool hc = HairCut;

bool hcl = HairColor;

bool hw = HairWashing;

if (hc == true)

{

time = time + 20;

}

if (hcl == true)

{

time = time + 60;

}

if (hw == true)

{

time = time + 10;

}

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Barbershop

{

class Program

{

static void Main(string[] args)

{

Progression pr1 = new Client(true, true, true);

pr1.TemplateMethod();

Progression pr2 = new Client(true, false, true);

pr2.TemplateMethod();

}

}

}

**Результат выполнения программы (рисунок 11):**

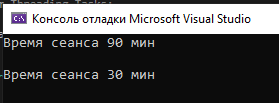


Рисунок 11 – Результат выполнения программы

**Вывод:**

Паттерн Шаблонный метод предлагает разбить алгоритм на последовательность шагов, описать эти шаги в отдельных методах и вызывать их в одном шаблонном методе друг за другом.

Был применен паттерн Шаблонный метод для проектирования сложного алгоритма процесса стрижки.

# Упражнение 7. Фасад (Facade)

Фасад (Facade) представляет структурный шаблон проектирования, который позволяет скрыть сложность системы с помощью предоставления упрощенного интерфейса для взаимодействия с ней путем сведения всех возможных внешних вызовов к одному объекту, делегирующему их соответствующим объектам системы.

В этом упражнении реализуется интерфейс высокого уровня сложной системы (ПО микроволновой печи), который упростит использование подсистемы. Для того, чтобы приготовить (разморозить) необходимо выполнить определённое количество различных действий, в определённой последовательности, при этом вращая платформу с продуктом. Если бы пользователю приходилось самому следить за каждым шагом процесса, то это было бы очень долго и неэффективно, поэтому на современных машинах достаточно выбрать нужную программу и нажать старт, после чего она сама сделает всё что необходимо.

Классы проекта:

* Drive;
* Power;
* Notification;
* Microwave;
* Program.

**Класс Drive:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Drive

{

public event EventHandler driveevent;

private string twist;

public string Twist

{

get { return twist; }

set

{

twist = value;

if (driveevent != null)

driveevent(this, new EventArgs());

}

}

public Drive()

{

Twist = "Исходная позиция";

}

public void TurlLeft()

{

Twist = "Поворот налево";

}

public void TurlRight()

{

Twist = "Поворот направо";

}

public void Stop()

{

Twist = "Стоп";

}

public override string ToString()

{

string s = String.Format("Привод: {0}", Twist);

return s;

}

}

}

**Класс Power:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Power

{

public event EventHandler powerevent;

private int \_power;

public int MicrowavePower

{

get { return \_power; }

set

{

\_power = value;

if (powerevent != null)

powerevent(this, new EventArgs());

}

}

public override string ToString()

{

string s = String.Format("Задана мощность {0}w ", MicrowavePower);

return s;

}

}

}

**Класс Notification:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Notification

{

public event EventHandler notificationevent;

private string mess;

public string MessageFin

{

get { return mess; }

set

{

mess = value;

if (notificationevent != null)

notificationevent(this, new EventArgs());

}

}

public void StartNotification()

{

MessageFin = "Операция началась";

}

public void StopNotification()

{

MessageFin = "Операция завершена";

}

public override string ToString()

{

string s = String.Format("Информация: {0}", MessageFin);

return s;

}

}

}

**Класс Microwave:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Microwave

{

private Drive \_drive;

private Power \_power;

private Notification \_notification;

public Microwave(Drive drive, Power power, Notification notification)

{

\_drive = drive;

\_power = power;

\_notification = notification;

}

public void Defrost()

{

\_notification.StartNotification();

\_power.MicrowavePower = 1000;

\_drive.TurlRight();

\_drive.TurlRight();

\_power.MicrowavePower = 500;

\_drive.Stop();

\_drive.TurlLeft();

\_drive.TurlLeft();

\_power.MicrowavePower = 200;

\_drive.Stop();

\_drive.TurlRight();

\_drive.TurlRight();

\_drive.Stop();

\_power.MicrowavePower = 0;

\_notification.StopNotification();

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Program

{

static void Main(string[] args)

{

var drive = new Drive();

var power = new Power();

var notification = new Notification();

var microwave = new Microwave(drive, power, notification);

power.powerevent += power\_powerevent;

drive.driveevent += drive\_driveevent;

notification.notificationevent += notification\_notificationevent;

Console.WriteLine("Разморозка");

microwave.Defrost();

}

static void notification\_notificationevent(object sender, EventArgs e)

{

Notification n = (Notification)sender;

Console.WriteLine(n.ToString());

}

static void drive\_driveevent(object sender, EventArgs e)

{

Drive d = (Drive)sender;

Console.WriteLine(d.ToString());

}

private static void power\_powerevent(object sender, EventArgs e)

{

Power p = (Power)sender;

Console.WriteLine(p.ToString());

}

}

}

**Результат выполнения программы (рисунок 12):**

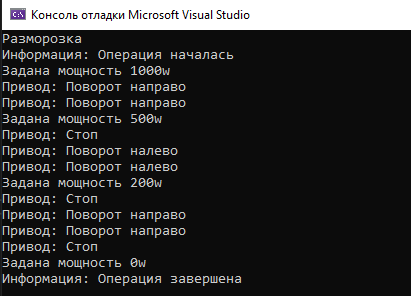


Рисунок 12 – Результат выполнения программы

**Контрольное задание:**

Добавлен в класс-фасад метод, реализующий приготовление продукта.

Добавленный метод:

* MediumRare.

**Класс Microwave (метод MediumRare):**

public void MediumRare()

{

\_notification.StartNotification();

\_power.MicrowavePower = 500;

\_drive.TurlRight();

\_power.MicrowavePower = 1000;

\_drive.TurlRight();

\_power.MicrowavePower = 1500;

\_drive.TurlRight();

\_drive.Stop();

\_power.MicrowavePower = 1000;

\_drive.TurlLeft();

\_power.MicrowavePower = 800;

\_drive.TurlLeft();

\_power.MicrowavePower = 400;

\_drive.TurlLeft();

\_drive.Stop();

\_power.MicrowavePower = 0;

\_notification.StopNotification();

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Facade

{

class Program

{

static void Main(string[] args)

{

var drive = new Drive();

var power = new Power();

var notification = new Notification();

var microwave = new Microwave(drive, power, notification);

power.powerevent += power\_powerevent;

drive.driveevent += drive\_driveevent;

notification.notificationevent += notification\_notificationevent;

/\*Console.WriteLine("Разморозка");

microwave.Defrost();\*/

Console.WriteLine("Приготовление продукта");

microwave.MediumRare();

}

static void notification\_notificationevent(object sender, EventArgs e)

{

Notification n = (Notification)sender;

Console.WriteLine(n.ToString());

}

static void drive\_driveevent(object sender, EventArgs e)

{

Drive d = (Drive)sender;

Console.WriteLine(d.ToString());

}

private static void power\_powerevent(object sender, EventArgs e)

{

Power p = (Power)sender;

Console.WriteLine(p.ToString());

}

}

}

**Результат выполнения программы (рисунок 13):**

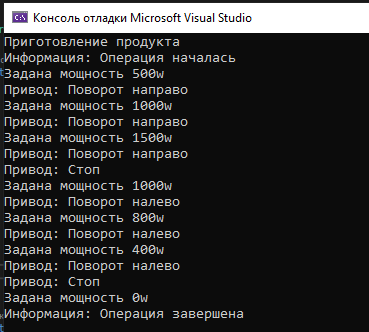


Рисунок 13 – Результат выполнения программы

**Вывод:**

Шаблон проектирования Фасад — это простой интерфейс для работы со сложной подсистемой, содержащей множество классов, удобный шаблон проектирования, скрывающий всю объемность системы от пользователя. Фасад удобен при использовании сложной библиотеки с множеством подвижных частей, когда необходимо работать только с определенной ее частью.

# Упражнение 8. Цепочка обязанностей (Chain of Responsibility)

Паттерн Chain of Responsibility – паттерн поведения, предоставляющий возможность обработать запрос нескольким объектам, тем самым устраняет возможность связывания отправителя запроса с его получателем. Все возможные обработчики запроса образуют цепочку, а сам запрос перемещается по этой цепочке, пока один из ее объектов не обработает запрос.

Каждый объект при получении запроса выбирает, либо обработать запрос, либо передать выполнение запроса следующему по цепочке.

В этом упражнении реализуется система отправки определенной суммы денег. Конкретный способ отправки неизвестен, так как может использоваться, например: банковский перевод, системы перевода типа WesternUnion и Unistream или система онлайн-платежей PayPal. Требуется просто внести сумму, выбрать адресата и нажать на кнопку.

Классы проекта:

* Program;
* Receiver;
* PaymentHandler;
* ConcretePaymentHandler.

**Класс Receiver:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace ChainofResponsibility

{

class Receiver

{

// банковские переводы

public bool BankTransfer { get; set; }

// денежные переводы - WesternUnion, Unistream

public bool MoneyTransfer { get; set; }

// перевод через PayPal

public bool PayPalTransfer { get; set; }

public Receiver(bool bt, bool mt, bool ppt)

{

BankTransfer = bt;

MoneyTransfer = mt;

PayPalTransfer = ppt;

}

}

}

**Класс PaymentHandler:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace ChainofResponsibility

{

abstract class PaymentHandler

{

public PaymentHandler Successor { get; set; }

public abstract void Handle(Receiver receiver);

}

}

**Класс ConcretePaymentHandler:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace ChainofResponsibility

{

class ConcretePaymentHandler

{

}

class BankPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.BankTransfer == true)

Console.WriteLine("Выполняем банковский перевод");

else if (Successor != null)

Successor.Handle(receiver);

}

}

class MoneyPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.MoneyTransfer == true)

Console.WriteLine("Выполняем перевод через системы денежных переводов");

else if (Successor != null)

Successor.Handle(receiver);

}

}

class PayPalPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.PayPalTransfer == true)

Console.WriteLine("Выполняем перевод через систему PayPal");

else if (Successor != null)

Successor.Handle(receiver);

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace ChainofResponsibility

{

class Program

{

static void Main(string[] args)

{

Receiver receiver = new Receiver(false, true, false);

PaymentHandler bankPaymentHandler = new BankPaymentHandler();

PaymentHandler moneyPaymentHnadler = new MoneyPaymentHandler();

PaymentHandler paypalPaymentHandler = new PayPalPaymentHandler();

paypalPaymentHandler.Successor = bankPaymentHandler;

bankPaymentHandler.Successor = moneyPaymentHnadler;

paypalPaymentHandler.Handle(receiver);

}

}

}

**Результат выполнения программы (рисунок 14):**



Рисунок 14 – Результат выполнения программы

**Вывод:**

Данный шаблон проектирования Chain of Responsibility – паттерн, предоставляющий возможность обработать запрос нескольким объектам, которые могут обрабатывать сообщения определенного типа, тем самым устраняет возможность связывания отправителя запроса с его получателем.

# Упражнение 9. Команда (Command)

В этом упражнении реализуется программный калькулятор с простыми арифметическими операциями и операциями отмены и повтора, используя шаблон команда – поведенческий шаблон проектирования, представляющий действие. Объект команды будет заключать в себе само действие и его параметры.

Состав программных блоков калькулятора включает:

* блок управления (ControlUnit) – организует работу калькулятора, выдавая в требуемый момент элементарные объекты-команды типа: Add, Sub, Mul, Div, Undo, Redo. При этом блок управления должен сохранять историю использования команд, а также отменять и восстанавливать ранее выполненные команды;
* арифметическое устройство (ArithmeticUnit), которое после получения «сигнала» (одной из четырех команд Add, Sub, Mul, Div) на вход выполняет арифметическую операцию;
* команды Add, Sub, Mul, Div – специальные объекты-команды, которые блок управления использует для управления арифметическим устройством. Каждый объект-команда связан с этим устройством и умеет им управлять.

Диаграмма классов изображена на рисунке 15.

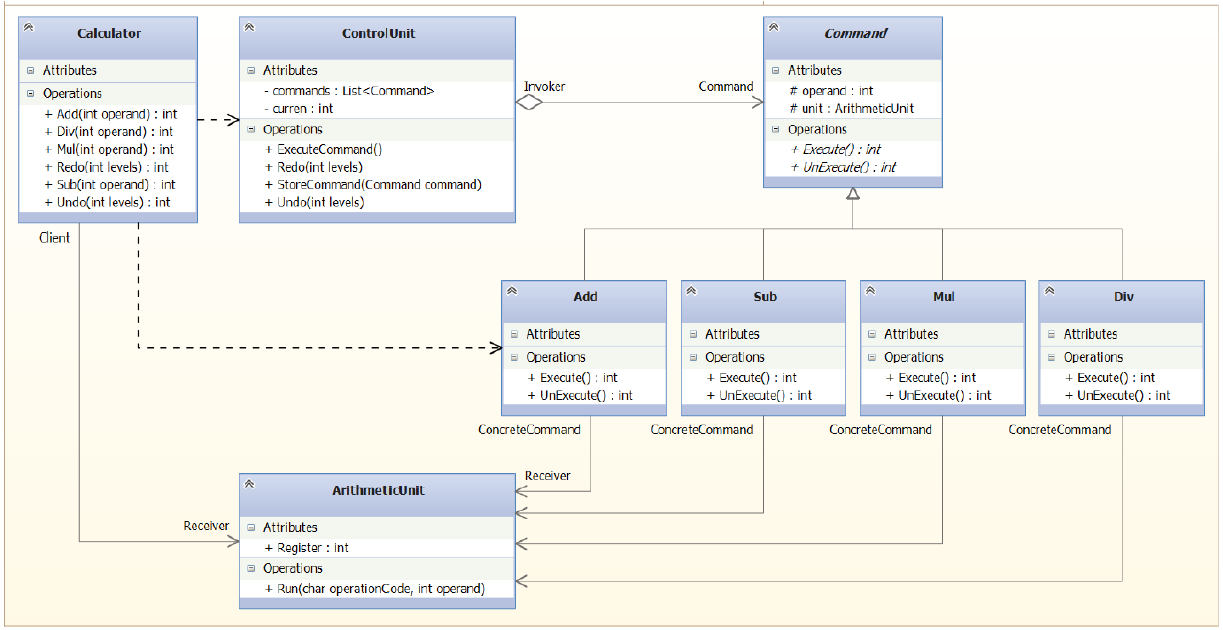


Рисунок 15 – Диаграмма классов шаблона «Команда»

Классы проекта:

* Program;
* Command;
* ArithmeticUnit;
* ControlUnit;
* Add;
* Calculator;
* Mul;
* Div;
* Sub.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Command

{

class Program

{

static void Main(string[] args)

{

var calculator = new Calculator();

double result = 0;

result = calculator.Add(15);

Console.WriteLine(result);

result = calculator.Add(2);

Console.WriteLine(result);

result = calculator.Add(8);

Console.WriteLine(result);

result = calculator.Sub(8);

Console.WriteLine(result);

result = calculator.Mul(500);

Console.WriteLine(result);

result = calculator.Sub(278);

Console.WriteLine(result);

result = calculator.Div(2);

Console.WriteLine(result);

}

}

}

**Класс Command:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Command

{

abstract class Command

{

protected ArithmeticUnit unit;

protected double operand;

public abstract void Execute();

public abstract void UnExecute();

}

}

**Класс ArithmeticUnit:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Command

{

class ArithmeticUnit

{

public double Register { get; private set; }

public void Run(char operationCode, double operand)

{

switch (operationCode)

{

case '+':

Register += operand;

break;

case '-':

Register -= operand;

break;

case '\*':

Register \*= operand;

break;

case '/':

Register /= operand;

break;

}

}

}

}

**Класс ControlUnit:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Command

{

class ControlUnit

{

private List<Command> commands = new List<Command>();

private int current = 0;

public void StoreCommand(Command command)

{

commands.Add(command);

}

public void ExecuteCommand()

{

commands[current].Execute();

current++;

}

public void Undo()

{

commands[current - 1].UnExecute();

}

public void Redo()

{

commands[current - 1].Execute();

}

}

}

**Класс Add:**

//Сложение

class Add : Command

{

public Add(ArithmeticUnit unit, double operand)

{

this.unit = unit;

this.operand = operand;

}

public override void Execute()

{

unit.Run('+', operand);

}

public override void UnExecute()

{

unit.Run('-', operand);

}

}

**Класс Calculator:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Command

{

class Calculator

{

ArithmeticUnit arithmeticUnit;

ControlUnit controlUnit;

public Calculator()

{

arithmeticUnit = new ArithmeticUnit();

controlUnit = new ControlUnit();

}

private double Run(Command command)

{

controlUnit.StoreCommand(command);

controlUnit.ExecuteCommand();

return arithmeticUnit.Register;

}

public double Add(double operand)

{

return Run(new Add(arithmeticUnit, operand));

}

public double Sub(double operand)

{

return Run(new Sub(arithmeticUnit, operand));

}

public double Div(double operand)

{

return Run(new Div(arithmeticUnit, operand));

}

public double Mul(double operand)

{

return Run(new Mul(arithmeticUnit, operand));

}

}

}

**Класс Mul:**

//Умножение

class Mul : Command

{

public Mul(ArithmeticUnit unit, double operand)

{

this.unit = unit;

this.operand = operand;

}

public override void Execute()

{

unit.Run('\*', operand);

}

public override void UnExecute()

{

unit.Run('/', operand);

}

}

**Класс Div:**

//Деление

class Div : Command

{

public Div(ArithmeticUnit unit, double operand)

{

this.unit = unit;

this.operand = operand;

}

public override void Execute()

{

unit.Run('/', operand);

}

public override void UnExecute()

{

unit.Run('\*', operand);

}

}

**Класс Sub:**

//Вычитание

class Sub : Command

{

public Sub(ArithmeticUnit unit, double operand)

{

this.unit = unit;

this.operand = operand;

}

public override void Execute()

{

unit.Run('-', operand);

}

public override void UnExecute()

{

unit.Run('+', operand);

}

}

**Результат выполнения программы (рисунок 16):**

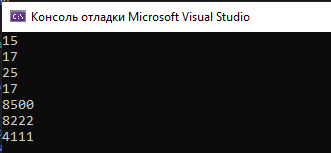


Рисунок 16 – Результат выполнения программы

**Вывод:**

С помощью паттерна Команда был реализован программный калькулятор с простыми арифметическими операциями и операциями отмены и повтора, используя шаблон команда – поведенческий шаблон проектирования, представляющий действие.

# Упражнение 10. Декоратор (Decorator)

Декоратор представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность, т.е. расширяет поведение объекта во время выполнения, добавляя или изменяя операции, которые будут осуществляться при обработке запроса.

Декоратор применяется в случае, когда нецелесообразно использование наследования. Например, если требуется определить множество различных функциональностей и для каждой функциональности наследовать отдельный класс, то структура классов может сильно разрастись, тем более еще больше она может разрастись, если необходимо создать классы, реализующие все возможные сочетания добавляемых функциональностей.

Данный шаблон требует общего интерфейса, реализовываемого классами декоратора, которые могут объединяться для обработки сложных бизнес-правил.

В этом упражнении реализуется шаблон Декоратор для следующей модели. В автосалоне продаются автомобили разной комплектации. Есть базовая комплектация, к которой по желанию клиента добавляются различные дополнения, влияющие на итоговую стоимость.

Требуется реализовать структуру классов, позволяющую динамически в процессе выполнения программы определять новые возможности объектов.

Классы проекта:

* Program;
* AutoBase;
* Renault;
* DecoratorOptions;
* MediaNAV;
* SystemSecurity.

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class Program

{

static void Main(string[] args)

{

Renault reno = new Renault("Рено", "Renault LOGAN Active", 499.0);

Print(reno);

AutoBase myreno = new MediaNAV(reno, "Навигация");

Print(myreno);

AutoBase newmyReno = new SystemSecurity(new MediaNAV(reno, "Навигация"),

"Безопасность");

Print(newmyReno);

}

private static void Print(AutoBase av)

{

Console.WriteLine(av.ToString());

}

}

}

**Класс AutoBase:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

public abstract class AutoBase

{

public string Name { get; set; }

public string Description { get; set; }

public double CostBase { get; set; }

public abstract double GetCost();

public override string ToString()

{

string s = String.Format("Ваш автомобиль: \n{0} \nОписание: {1} \nСтоимость {2}\n", Name, Description, GetCost());

return s;

}

}

}

**Класс Renault:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class Renault : AutoBase

{

public Renault(string name, string info, double costbase)

{

Name = name;

Description = info;

CostBase = costbase;

}

public override double GetCost()

{

return CostBase \* 1.18;

}

}

}

**Класс DecoratorOptions:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

abstract class DecoratorOptions : AutoBase

{

public AutoBase AutoProperty { protected get; set; }

public string Title { get; set; }

public DecoratorOptions(AutoBase au, string tit)

{

AutoProperty = au;

Title = tit;

}

public override double GetCost()

{

return CostBase;

}

}

}

**Класс MediaNAV:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class MediaNAV : DecoratorOptions

{

public MediaNAV(AutoBase p, string t) : base(p, t)

{

AutoProperty = p;

Name = p.Name + ". Современный";

Description = p.Description + ". " + this.Title + ". Обновленная мультимедийная навигационная система";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 15.99;

}

}

}

**Класс SystemSecurity:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class SystemSecurity : DecoratorOptions

{

public SystemSecurity(AutoBase p, string t) : base(p, t)

{

AutoProperty = p;

Name = p.Name + ". Повышенной безопасности";

Description = p.Description + ". " + this.Title + ". Передние боковые подушки безопасности, ESP -система динамической стабилизации автомобиля";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 20.99;

}

}

}

**Результат выполнения программы (рисунок 17):**

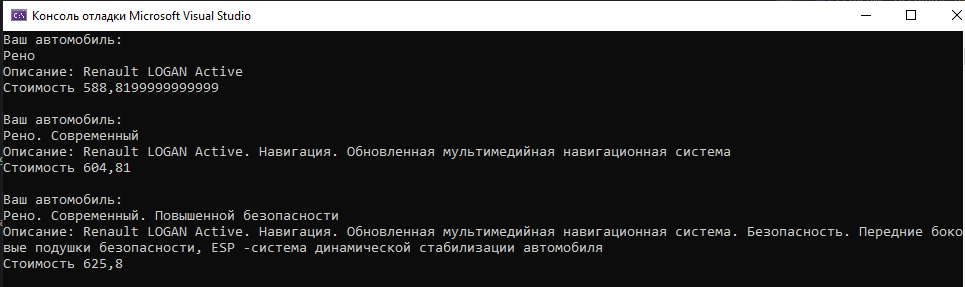


Рисунок 17 – Результат выполнения программы

**Контрольное задание:**

В разработанное приложение добавлен класс для нового автомобиля и новая функциональная возможность.

Добавленные классы:

* Lamborghini;
* AutoAppearance;
* Program.

**Класс Lamborghini:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class Lamborghini : AutoBase

{

public Lamborghini(string name, string info, double costbase)

{

Name = name;

Description = info;

CostBase = costbase;

}

public override double GetCost()

{

return CostBase \* 1.95;

}

}

}

**Класс AutoAppearance:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class AutoAppearance : DecoratorOptions

{

public AutoAppearance(AutoBase p, string t) : base(p, t)

{

AutoProperty = p;

Name = p.Name + "Extarier";

Description = p.Description + ". " + this.Title + ". Темный стиль";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 320.99;

}

}

}

**Класс Program:**

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

using System.Threading.Tasks;

namespace Decorator

{

class Program

{

static void Main(string[] args)

{

Renault reno = new Renault("Рено", "Renault LOGAN Active", 499.0);

Print(reno);

AutoBase myreno = new MediaNAV(reno, "Навигация");

Print(myreno);

AutoBase newmyReno = new SystemSecurity(new MediaNAV(reno, "Навигация"), "Безопасность");

Print(newmyReno);

Lamborghini lamborghini = new Lamborghini("Lamborghini ", "Lamborghini Aventador", 79999.0);

Print(lamborghini);

AutoBase mylamborghini = new AutoAppearance(lamborghini, "Внешность ");

Print(mylamborghini);

AutoBase newmylamborghini = new SystemSecurity(new AutoAppearance(lamborghini, "Appearance "), "Security ");

Print(newmylamborghini);

}

private static void Print(AutoBase av)

{

Console.WriteLine(av.ToString());

}

}

}

**Результат выполнения программы (рисунок 18):**

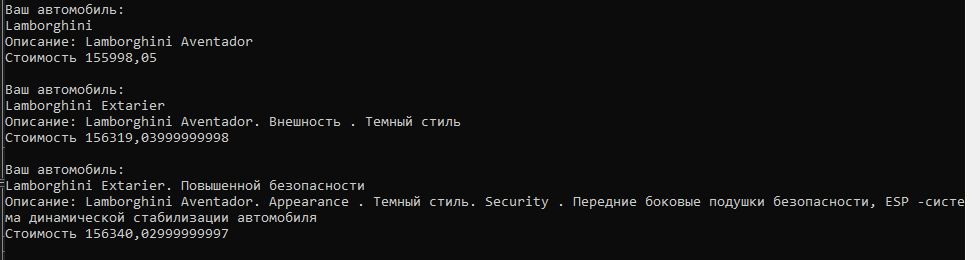


Рисунок 18 – Результат выполнения программы

**Вывод:**

Шаблон проектирования Декоратор представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность, т.е. расширяет поведение объекта во время выполнения, добавляя или изменяя операции, которые будут осуществляться при обработке запроса.

# Литература

1. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2010. – 368 с.
2. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C#. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 768 с.
3. Тепляков С. Паттерны проектирования на платформе .NET. — СПб.: Питер, 2015. – 320 с.
4. Шевчук А., Охрименко Д., Касьянов А. Design Patterns via C#. Приемы объектно-ориентированного проектирования. ITVDN.com, 2015. – 288с.
5. Шаблоны проектирования GoF [Электронный ресурс] Режим доступа: Шаблоны проектирования\_GoF\_руководство.pdf.