Министерство общего и профессионального образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**«Санкт-Петербургский национальный исследовательский университет ИТМО»**

Изображение выглядит как игра, стол

Автоматически созданное описание

Практическая работа 4.

Построение и исследование программной системы на основе шаблонов проектирования

По дисциплине: Проектирование инфокоммуникационных систем

Выполнил:

Студентка гр. K4111с

Бедняков Евгений Федорович

Проверил:

Волкодав Владимир Алексеевич

Санкт-Петербург

2020

Содержание

[Упражнение 1. Адаптер 3](#_Toc51435167)

[Упражнение 2. Абстрактная фабрика 11](#_Toc51435168)

[Упражнение 3. Фабричный метод. 19](#_Toc51435169)

[Упражнение 4. Одиночка. 26](#_Toc51435170)

[Упражнение 5. Стратегия. 29](#_Toc51435171)

[Упражнение 6. Шаблонный метод. Template Method. 36](#_Toc51435172)

[Упражнение 7. Фасад. 42](#_Toc51435173)

[Упражнение 8. Цепочка обязанностей. Chain of Responsibility. 51](#_Toc51435174)

[Упражнение 9. Команда. Command. 54](#_Toc51435175)

[Упражнение 10. Декоратор. Decorator. 61](#_Toc51435176)

# Упражнение 1. Адаптер

Классы проекта:

* Program
* Gamer
* AdapterGame
* Monet
* Kost

Интерфейсы проекта:

* IGame

**Интерфейс IGame:**

namespace Adapter

{

interface IGame

{

int Brosok();

}

}

**Класс Gamer:**

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 Monet:**

class Monet

{

Random r;

public Monet()

{

r = new Random();

}

public int BrosokM()

{

//Random number from 1 to 2.

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

return res;

}

}

**Класс** **AdapterGame:**

class AdapterGame: IGame

{

Monet mot;

public AdapterGame(Monet mt)

{

mot = mt;

}

public int Brosok()

{

return mot.BrosokM();

}

}

**Класс Kost:**

class Kost : IGame

{

Random r;

public Kost()

{

r = new Random();

}

public int Brosok()

{

//Random number from 1 to 6.

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

return res;

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

Kost kubik = new Kost();

Gamer g1 = new Gamer("Ivan");

Console.WriteLine("You have {0} points for player {1}", g1.SeansGame(kubik), g1.ToString());

Monet mon = new Monet();

IGame bmon = new AdapterGame(mon);

Console.WriteLine("You have {0} for player {1}", g1.SeansGame(bmon), g1.ToString());

}

}

Результат работы программы:

Изображение выглядит как снимок экрана, рисунок

Автоматически созданное описание

**Контрольное задание:**

Классы проекта:

* Program
* MeasureFaringeit
* MeasureCelsiy
* Measure
* Adapter

Интерфейсы проекта:

* IGrad

**Интерфейс IGrad:**

interface IGrad

{

int Grad();

}

**Класс MeasureFaringeit:**

class MeasureFaringeit : IGrad

{

Random r;

public MeasureFaringeit()

{

r = new Random();

}

public int Grad()

{

//Fandom number

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

return res;

}

}

**Класс MeasureCelsiy:**

class MeasureCelsiy

{

Random r;

public MeasureCelsiy()

{

r = new Random();

}

public int GradC()

{

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

return res;

}

}

**Класс Measure:**

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 Adapter:**

class Adapter : IGrad

{

MeasureCelsiy cel;

public Adapter(MeasureCelsiy c)

{

cel = c;

}

public int Grad()

{

return cel.GradC();

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

MeasureFaringeit far = new MeasureFaringeit();

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

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

MeasureCelsiy cel = new MeasureCelsiy();

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

IGrad measCel = new Adapter(cel);

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

}

}

Результат выполнения программы:

Изображение выглядит как рисунок

Автоматически созданное описание

**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

Классы проекта:

* Program
* CarFactory
* AbstractCar
* AbstractEngine
* AbstractKuzov
* FordFactory
* FordCar
* FordEngine
* Client

**Класс CarFactory:**

abstract class CarFactory

{

public abstract AbstractCar CreateCar();

public abstract AbstractEngine CreateEngine();

public abstract AbstractKuzov CreateKuzov();

}

**Класс AbstractCar:**

abstract class AbstractCar

{

public string Name { get; set; }

public abstract int MaxSpeed(AbstractEngine engine);

public abstract string Kuzov(AbstractKuzov kuzov);

}

**Класс AbstractEngine:**

abstract class AbstractEngine

{

public int max\_speed { get; set; }

}

**Класс AbstractKuzov:**

abstract class AbstractKuzov

{

public string kuzov { get; set; }

}

**Класс FordFactory:**

class FordFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new FordCar("Ford");

}

public override AbstractEngine CreateEngine()

{

return new FordEngine();

}

public override AbstractKuzov CreateKuzov()

{

return new FordKuzov();

}

}

**Класс** **FordCar:**

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 Kuzov(AbstractKuzov kuzov)

{

string kuz = kuzov.kuzov;

return kuz;

}

public override string ToString()

{

return "Car " + Name;

}

}

**Класс FordEngine:**

class FordEngine : AbstractEngine

{

public FordEngine()

{

max\_speed = 220;

}

}

**Класс Client:**

class Client

{

private AbstractCar abstractCar;

private AbstractEngine abstractEngine;

private AbstractKuzov abstractkuzov;

public Client(CarFactory car\_factory)

{

abstractCar = car\_factory.CreateCar();

abstractEngine = car\_factory.CreateEngine();

abstractkuzov = car\_factory.CreateKuzov();

}

public int RunMaxSpeed()

{

return abstractCar.MaxSpeed(abstractEngine);

}

public string KuzovAuto()

{

return abstractkuzov.kuzov;

}

public override string ToString()

{

return abstractCar.ToString();

}

}

**Класс Program:**

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/hour", c1.ToString(), c1.RunMaxSpeed());

}

}

Результат работы программы:

**Изображение выглядит как рисунок

Автоматически созданное описание**

**Контрольное задание:**

Добавленные классы:

* AbstractKuzov
* AudiCar
* AudiEng
* AudiFactory
* AudiKuzov

**Класс AudiCar:**

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 Kuzov(AbstractKuzov kuzov)

{

string k = kuzov.kuzov;

return k;

}

public override string ToString()

{

return "Car - " + Name;

}

}

**Класс AudiEng:**

class AudiEng : AbstractEngine

{

public AudiEng()

{

max\_speed = 310;

}

}

**Класс AudiFactory:**

class AudiFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new AudiCar("Audi");

}

public override AbstractEngine CreateEngine()

{

return new AudiEng();

}

public override AbstractKuzov CreateKuzov()

{

return new AudiKuzov();

}

}

**Класс AudiKuzov:**

class AudiKuzov : AbstractKuzov

{

public AudiKuzov()

{

kuzov = "A5";

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

CarFactory ford\_car = new FordFactory();

Client c1 = new Client(ford\_car);

Console.WriteLine("Kuzov: {0} Max speed {1} is {2} km/hour", c1.KuzovAuto(), c1.ToString(), c1.RunMaxSpeed());

CarFactory audi\_car = new AudiFactory();

Client c2 = new Client(audi\_car);

Console.WriteLine("Kuzov: {0} Max speed {1} is {2} km/hour ", c2.KuzovAuto(), c2.ToString(), c2.RunMaxSpeed());

}

}

Результат выполнения программы:

Изображение выглядит как рисунок

Автоматически созданное описание

**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.

Классы проекта:

* Program
* TransportService
* TransportCompany
* TaxiServices
* Shipping
* TaxiTransCom
* TaxiTransCom

**Класс TransportService:**

abstract class TransportService

{

public string Name { get; set; }

public TransportService(string name)

{

Name = name;

}

abstract public double CostTransportation(double distance);

}

**Класс TransportCompany:**

abstract class TransportCompany

{

public string Name { get; set; }

public TransportCompany(string n)

{

Name = n;

}

public override string ToString()

{

return Name;

}

//Fabric Method

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

}

**Класс TaxiServices:**

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("Firm {0}, the category of a ride {1}", Name, Category);

return s;

}

}

**Класс Shipping:**

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("Firm {0}, cost on tariff {1}", Name, Tariff);

return s;

}

}

**Класс TaxiTransCom:**

class TaxiTransCom : TransportCompany

{

public TaxiTransCom(string name) : base(name)

{ }

public override TransportService Create(string n, int c)

{

return new TaxiServices(Name, c);

}

}

**Класс ShipTransCom:**

class ShipTransCom : TransportCompany

{

public ShipTransCom(string name) : base(name)

{ }

public override TransportService Create(string n, int t)

{

return new Shipping(Name, t);

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

TransportCompany trCom = new TaxiTransCom("Taxi Service");

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

double dist = 15.5;

Print(compService, dist);

TransportCompany gCom = new ShipTransCom("Shipping Service");

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

double distg = 150.5;

Print(compService, distg);

}

private static void Print(TransportService compTax, double distg)

{

Console.WriteLine("Company {0}, distance {1}, cost: {2} ",

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

}

}

Результат работы программы:

**Изображение выглядит как снимок экрана

Автоматически созданное описание**

**Контрольное задание:**

Добавленные классы:

* AlcDriver
* AlcDriverCom

**Класс AlcDriver:**

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}, доставка по тарифу {1}",

Name, Tariff);

return s;

}

}

**Класс AlcDriverCom:**

class AlcDriverCom : TransportCompany

{

public AlcDriverCom(string name) : base(name) { }

public override TransportService Create(string n, int k)

{

return new AlcDriver(Name, k);

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

TransportCompany trCom = new TaxiTransCom("Taxi Service");

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

double dist = 15.5;

Print(compService, dist);

TransportCompany gCom = new ShipTransCom("Shipping Service");

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

double distg = 150.5;

Print(compService, distg);

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

compService = alcCom.Create("Пьяный водитель", 3);

double distal = 100.5;

Print(compService, distal);

}

private static void Print(TransportService compTax, double distg)

{

Console.WriteLine("Company {0}, distance {1}, cost: {2} ",

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

}

}

Результат выполнения программы:

**Изображение выглядит как снимок экрана

Автоматически созданное описание**

**Вывод:**

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

# Упражнение 4. Одиночка.

Классы проекта:

* Log
* Operation
* Program

**Класс Log:**

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();

}

}

public static void Loger(string logMessage, TextWriter w)

{

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

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

DateTime.Now.ToLongDateString());

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

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

}

}

**Класс Operation:**

class Operation

{

public static double Run(char operationCode, int operand)

{

Log lg2 = Log.MyLog;

double rez = 0;

switch (operationCode)

{

case '+':

rez += operand;

lg2.LogExecution("Slozhenie " + operand);

break;

case '-':

rez -= operand;

lg2.LogExecution("Vichitanie " + operand);

break;

case '\*':

rez \*= operand;

break;

case '/':

case ':':

rez /= operand;

break;

}

return rez;

}

}

**Класс Program:**

class Program

{

static void Main(string[] args)

{

Log lg = Log.MyLog;

lg.LogExecution("Method Main() ");

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

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

}

}

Результат работы программы:

Изображение выглядит как птица, дерево

Автоматически созданное описание

**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.

Классы проекта:

* Program
* StrategySort
* InsertionSort
* SelectionSort
* Context

**Класс StrategySort:**

abstract class StrategySort

{

public string Title { get; set; }

public abstract void Sort(int[] array);

}

**Класс InsertionSort:**

class InsertionSort : StrategySort

{

public InsertionSort()

{

Title = "Insertion\_Sort";

}

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:**

class SelectionSort : StrategySort

{

public SelectionSort()

{

Title = "Selection\_Sort";

}

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:**

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();

}

}

**Класс Program:**

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);

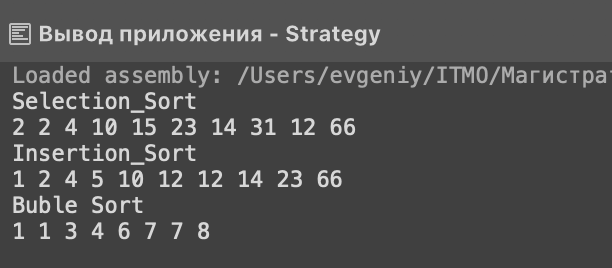
context.Sort();

context.PrintArray();

}

}

Результат работы программы:



**Контрольное задание:**

Добавленный класс:

* BubleSort

**Класс BubleSort:**

class BubleSort : StrategySort

{

public BubleSort()

{

Title = "Пузырьковая сортировка";

}

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:**

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 BubleSort();

context = new Context(sort, arr3);

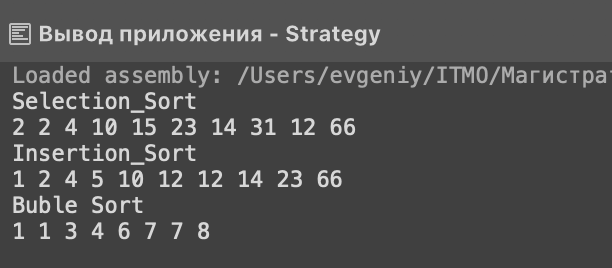
context.Sort();

context.PrintArray();

}

}

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.

Классы проекта:

* Program
* Progression
* ArithmeticProgression
* GeometricProgression

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("Sequense: ");

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();

}

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);

}

}

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);

}

}

class Program

{

static void Main(string[] args)

{

Progression val = new ArithmeticProgression(1, 20, 3);

val.TemplateMethod();

Progression geom = new GeometricProgression(1, 100, 3);

geom.TemplateMethod();

}

}

Результат работы программы:

Изображение выглядит как снимок экрана, рисунок

Автоматически созданное описание

**Контрольное задание:**

static void Main(string[] args)

{

int f;

Console.WriteLine("Длина");

f = Int32.Parse(Console.ReadLine());

Strizhka val = new Dlinat(f);

val.TemplateMethod();

}

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;

}

private void TemplateMethod()

{

InitProgression(HairCut, HairColor, HairWashing);

Progress();

Print(time);

}

private void Print(int time)

{

Console.WriteLine("Session time {0} min", time);

Console.WriteLine();

}

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

{

HairCut = a;

HairColor = b;

HairWashing = c;

}

public abstract void Progress();

}

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;

}

}

}

Результат работы программы:

Изображение выглядит как снимок экрана, рисунок

Автоматически созданное описание

**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.

Применили паттерн Шаблонный метод для проектирования сложного алгоритма процесса стрижки.

# Упражнение 7. Фасад.

Классы проекта:

* Drive
* Power
* Notification
* Microwave
* Program

**Класс Drive:**

class Drive

{

public event EventHandler driveevent;

public string twist;

public string Twist

{

get { return twist; }

set

{

twist = value;

if (driveevent != null)

driveevent(this, new EventArgs());

}

}

public Drive()

{

Twist = "Start posision";

}

public void TurlLeft()

{

Twist = "Turn left";

}

public void TurlRight()

{

Twist = "Turn Right";

}

public void Stop()

{

Twist = "Stop";

}

public override string ToString()

{

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

return s;

}

}

**Класс Power:**

class Power

{

public event EventHandler powerevent;

private int \_power;

public int MicrowaverPower

{

get { return \_power; }

set

{

\_power = value;

if (powerevent != null)

powerevent(this, new EventArgs());

}

}

public override string ToString()

{

string s = String.Format("Enter the power {0}w ", MicrowaverPower);

return s;

}

}

**Класс Notification:**

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 = "Operation started";

}

public void StopNotification()

{

MessageFin = "Operation finished";

}

public override string ToString()

{

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

return s;

}

}

**Класс Microwave:**

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.MicrowaverPower = 1000;

\_drive.TurlRight();

\_drive.TurlRight();

\_power.MicrowaverPower = 500;

\_drive.Stop();

\_drive.TurlLeft();

\_drive.TurlLeft();

\_power.MicrowaverPower = 200;

\_drive.Stop();

\_drive.TurlRight();

\_drive.TurlRight();

\_drive.Stop();

\_power.MicrowaverPower = 0;

\_notification.StopNotification();

}

}

**Класс Program:**

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("Defrost");

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:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Контрольное задание:**

Добавленный метод:

* MediumRare

**Метод MediumRare (класс Microwave):**

public void MediumRare()

{

\_notification.StartNotification();

\_power.MicrowaverPower = 500;

\_drive.TurlRight();

\_power.MicrowaverPower = 1000;

\_drive.TurlRight();

\_power.MicrowaverPower = 1500;

\_drive.TurlRight();

\_drive.Stop();

\_power.MicrowaverPower = 1000;

\_drive.TurlLeft();

\_power.MicrowaverPower = 800;

\_drive.TurlLeft();

\_power.MicrowaverPower = 400;

\_drive.TurlLeft();

\_drive.Stop();

\_power.MicrowaverPower = 0;

\_notification.StopNotification();

}

**Класс Program:**

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("Defrost");

microwave.Defrost();\*/

Console.WriteLine("Medium Rare");

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());

}

}

Результаты выполнения программы:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**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.

Классы проекта:

* Program
* Receiver
* PaymentHandler
* BankPaymentHandler
* MoneyPaymentHandler
* PayPalPaymentHandler

class Program

{

static void Main(string[] args)

{

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

PaymentHandler bankPaymentHandler = new BankPaymentHandler();

PaymentHandler moneyPaymentHnadler = new MoneyPaymentHandler();

PaymentHandler paypalPaymentHandler = new PayPalPaymentHandler();

bankPaymentHandler.Successor = paypalPaymentHandler;

paypalPaymentHandler.Successor = moneyPaymentHnadler;

bankPaymentHandler.Handle(receiver);

Console.ReadKey();

}

}

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;

}

}

abstract class PaymentHandler

{

public PaymentHandler Successor { get; set; }

public abstract void Handle(Receiver receiver);

}

class BankPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.BankTransfer == true)

Console.WriteLine("Bank transfer in process");

else if (Successor != null)

Successor.Handle(receiver);

}

}

class MoneyPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.MoneyTransfer == true)

Console.WriteLine("Money transfer in process");

else if (Successor != null)

Successor.Handle(receiver);

}

}

class PayPalPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.PayPalTransfer == true)

Console.WriteLine("PayPal transfer in process");

else if (Successor != null)

Successor.Handle(receiver);

}

}

Результаты выполнения программы:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**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.

Классы проекта:

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

class Program

{

static void Main(string[] args)

{

var calculator = new Calculator();

double result = 0;

result = calculator.Add(5);

Console.WriteLine(result);

result = calculator.Add(4);

Console.WriteLine(result);

result = calculator.Add(3);

Console.WriteLine(result);

Console.WriteLine("Контрольное задание:");

result = calculator.Mul(4);

Console.WriteLine(result);

result = calculator.Div(2);

Console.WriteLine(result);

result = calculator.Sub(1);

Console.WriteLine(result);

Console.ReadKey();

}

}

abstract class Command

{

protected ArithmeticUnit unit;

protected double operand;

public abstract void Execute();

public abstract void UnExecute();

}

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;

}

}

}

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();

}

}

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);

}

}

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);

}

}

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);

}

}

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);

}

}

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 Add(arithmeticUnit, operand));

}

public double Mul(double operand)

{

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

}

public double Div(double operand)

{

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

}

public double Redo(double operand)

{

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

}

public double Undo(double operand)

{

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

}

}

Результаты выполнения программы:

Изображение выглядит как снимок экрана, рисунок

Автоматически созданное описание

**Вывод:**

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

# Упражнение 10. Декоратор. Decorator.

Классы проекта:

* Program
* AutoBase
* Renault
* DecoratorOptions
* MediaNAV
* SystemSecurity

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("Your car: \n{0} \n discription: {1} \n Cost {2}\n",

Name, Description, GetCost());

return s;

}

}

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;

}

}

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;

}

}

class MediaNAV : DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Contemporary";

Description = p.Description + ". " + this.Title + ". Refresh multimedia NAV system";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 15.99;

}

}

class SystemSecurity :DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Higher security";

Description = p.Description + ". " + this.Title +

". Front side security pillow, ESP - dinamic stabilisation system";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 20.99;

}

}

class Program

{

static void Main(string[] args)

{

Renault reno = new Renault("RENAULT", "Renault Logan Active", 499.0);

Print(reno);

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

Print(myreno);

AutoBase newmyReno = new SystemSecurity(new MediaNAV(reno, "Navigation"), "Security");

Print(newmyReno);

}

private static void Print(AutoBase av)

{

Console.WriteLine(av.ToString());

}

}

Результат работы программы:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Контрольное задание:**

Добавленные классы:

* Audi
* AutoAppearance

class Audi : AutoBase

{

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

{

Name = name;

Description = info;

CostBase = costbase;

}

public override double GetCost()

{

return CostBase \* 1.30;

}

}

class AutoAppearance : DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + "Внешняя отделка автомобиля";

Description = p.Description + ". " + this.Title +

". Черный стайлинг-пакет внешней отделки";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 649.99;

}

}

class Program

{

static void Main(string[] args)

{

Audi audi = new Audi("Audi ", "Audi S5 Sportback ", 69999.0);

Print(audi);

AutoBase myaudi = new AutoAppearance(audi, "Appearance ");

Print(myaudi);

AutoBase newmyaudi = new SystemSecurity(new AutoAppearance(audi, "Appearance "), "Security ");

Print(newmyaudi);

}

private static void Print(AutoBase av)

{

Console.WriteLine(av.ToString());

}

}

Результат выполнения программы:

Изображение выглядит как снимок экрана

Автоматически созданное описание

**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.