Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

**ОТЧЕТ**

по Практической работе 4

«Построение и исследование программной системы на основе шаблонов проектирования»

Выполнил:

cтудент гр. K4113c

Гулько Георгий Сергеевич

Проверил:

к.т.н., доцент факультета ИКТ

Осипов Никита Александрович

Санкт-Петербург

2020 г.

**Задание:**

1. Изучить шаблоны проектирования GoF [1-4], определить особенности их

применения в программных системах.

2. Выполнить упражнения и контрольные задания руководства [5].

**Теоретическая часть:**

1. Абстрактная фабрика

*Назначение:* абстрактная фабрика предоставляет интерфейс для создания семейства взаимосвязанных или родственных объектов (dependent or related objects), не специфицируя их конкретных классов.

*Следует использовать:*

* Когда система не должна зависеть от способа создания новых объектов
* Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными

1. Адаптер

*Назначение:* преобразует интерфейс одного класса в интерфейс другого, который ожидают клиенты. Адаптер делает возможной совместную работу классов с несовместимыми интерфейсами.

*Следует использовать:*

* Когда необходимо использовать имеющийся класс, но его интерфейс не соответствует потребностям бизнесс логики.
* Когда надо использовать уже существующий класс совместно с другими классами, интерфейсы которых не совместимы.

1. Фабричный метод

*Назначение:* определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать инстанцирование подклассам.

*Следует использовать:*

* Когда заранее неизвестно, объекты каких типов необходимо создавать;
* Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать;
* Когда создание новых объектов необходимо делегировать из базового класса классам наследникам;

1. Одиночка

*Назначение:* гарантирует, что у класса есть только один экземпляр, и предоставляет глобальную точку доступа к нему.

*Следует использовать:*

Практически в любом приложении возникает необходимость в глобальных переменных или объектах с ограниченным числом экземпляров. Самый простой способ решить эту задачу — создать глобальный объект, который будет доступен из любой точки приложения. По своему определению синглтон гарантирует, что у некоего класса есть лишь один экземпляр. В некоторых случаях анализ предметной области строго требует, чтобы класс существовал лишь в одном экземпляре. Однако на практике паттерн «Синглтон» обычно используется для обеспечения доступа к какому-либо ресурсу, который требуется разным частям приложения.

1. Стратегия

*Назначение:* определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми. Стратегия позволяет изменять алгоритмы независимо от клиентов, которые ими пользуются.

*Следует использовать:*

* Когда есть несколько схожих классов , которые отличаются поведением. Можно задать один основной класс, а разные варианты поведения вынести в отдельные классы и при необходимости их применять;
* Когда необходимо обеспечить выбор из нескольких вариантов решений, которые можно легко менять в зависимости от условий;
* Когда необходимо менять поведение классов и объектов на стадии выполнения программы;
* Когда класс, применяющий определенную функциональность, ничего не должен знать о ее реализации

1. Шаблонный метод

*Назначение:* шаблонный метод определяет основу алгоритма и позволяет подклассам переопределять некоторые шаги алгоритма, не изменяя его структуры в целом. Шаблонный метод — это каркас, в который наследники могут подставить реализации недостающих элементов.

*Следует использовать:*

* Когда планируется, что в будущем подклассы должны будут переопределять различные этапы алгоритма без изменения его структуры
* Когда в классах, реализующим схожий алгоритм, происходит дублирование кода. Вынесение общего кода в шаблонный метод уменьшит его дублирование в подклассах.

1. Фасад

*Назначение:* предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистемы. Шаблон Фасад объединяет группу объектов в рамках одного специализированного интерфейса и переадресует вызовы его методов к этим объектам.

*Следует использовать:*

* Когда имеется сложная система, и необходимо упростить с ней работу. Фасад позволит определить одну точку взаимодействия между клиентом и системой.
* Когда надо уменьшить количество зависимостей между клиентом и сложной системой. Фасадные объекты позволяют отделить, изолировать компоненты системы от клиента и развивать и работать с ними независимо.
* Когда нужно определить подсистемы компонентов в сложной системе. Создание фасадов для компонентов каждой отдельной подсистемы позволит упростить взаимодействие между ними и повысить их независимость друг от друга.

1. Цепочка обязанностей

*Назначение:* позволяет избежать привязки отправителя запроса к его получателю, давая шанс обработать запрос нескольким объектам. Связывает объекты-получатели в цепочку и передает запрос вдоль этой цепочки, пока его не обработают. «Цепочка обязанностей» является довольно распространенным паттерном в .NET Framework, хотя не все знают, что часто пользуются им. Цепочка обязанностей — это любое событие, аргументы которого позволяют уведомить инициатора, что событие обработано с помощью метода Handle() или путем установки свойства Handled в True.

*Следует использовать:*

* Когда имеется более одного объекта, который может обработать определенный запрос;
* Когда надо передать запрос на выполнение одному из нескольких объект, точно не определяя, какому именно объекту;
* Когда набор объектов задается динамически.

1. Команда

*Назначение:* инкапсулирует запрос как объект, позволяя тем самым задавать параметры клиентов для обработки соответствующих запросов, ставить запросы в очередь или протоколировать их, а также поддерживать отмену операций.  
Паттерн «Команда» позволяет спрятать действие в объекте и отвязать источник этого действия от места его исполнения. Классический пример — проектирование пользовательского интерфейса. Пункт меню не должен знать, что происходит при его активизации пользователем, он должен знать лишь о некотором действии, которое нужно выполнить при нажатии кнопки.

*Следует использовать:*

* Когда необходимо обеспечить выполнение очереди запросов, а также их возможную отмену.
* Когда надо поддерживать логгирование изменений в результате запросов. Использование логов может помочь восстановить состояние системы - для этого необходимо будет использовать последовательность запротоколированных команд.
* Когда необходимо параметризировать объекты выполняемым действием, ставить запросы в очередь или поддерживать операции отмены (undo) и повтора (redo) действий.

1. Декоратор

*Назначение:* динамически добавляет объекту новые обязанности. Является гибкой альтернативой порождению подклассов с целью расширения функциональности.

*Следует использовать:*

* Когда надо динамически добавлять к объекту новые функциональные возможности. При этом данные возможности могут быть сняты с объекта
* Когда применение наследования неприемлемо. Например, если нам надо определить множество различных функциональностей и для каждой функциональности наследовать отдельный класс, то структура классов может очень сильно разрастись. Еще больше она может разрастись, если нам необходимо создать классы, реализующие все возможные сочетания добавляемых функциональностей.

**Ход работы:**

Упражнение 1. Адаптер (Adapter)

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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Adapter

{

class Program

{

static void Main(string[] args)

{

Kost kubik = new Kost();

Gamer g1 = new Gamer("Иван");

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());

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Adapter

{

interface IGame

{

int Brosok();

}

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;

}

}

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();

}

}

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; } }

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

return res;

}

}

class AdapterGame : IGame

{

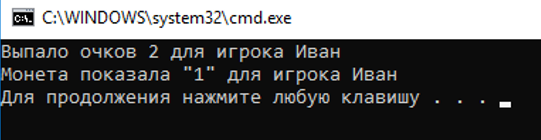
Monet mot;

public AdapterGame(Monet mt) { mot = mt; }

public int Brosok() { return mot.BrosokM(); }

}

}



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. Функциональность классов разработайте на свое усмотрение.

*Код задания:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Цельсий

{

class Program

{

static void Main(string[] args)

{

Cel c = new Cel();

System s = new System(c);

int res = s.Far();

Console.WriteLine("Градусы: {0}",res);

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Цельсий

{

interface Int

{

int Far();

}

class Cel

{

int r;

public Cel()

{

Console.WriteLine("Введите фаренгейты:");

r = int.Parse(Console.ReadLine());

}

public int Fromfartocel()

{

int res = (r - 32) \* 5 / 9;

return res;

}

}

class System : Int

{

Cel grad;

public System(Cel gr)

{

grad = gr;

}

public int Far()

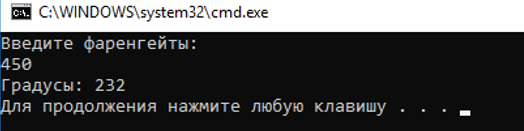
{

return grad.Fromfartocel();

}

}

}



Упражнение 2. Абстрактная фабрика (Abstract Factory)

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

*Код упражнения:*

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("Максимальная скорость {0} составляет {1} км/час, тип кузова: {2}", c1.ToString(), c1.RunMaxSpeed(), c1.RunCarType());

CarFactory audi\_car = new AudiFactory();

Client c2 = new Client(audi\_car);

Console.WriteLine("Максимальная скорость {0} составляет {1} км/час, тип кузова: {2}", c2.ToString(), c2.RunMaxSpeed(), c2.RunCarType());

CarFactory pego\_car = PegoFactory.MyPegoFactory;

Client c3 = new Client(pego\_car);

Console.WriteLine("Максимальная скорость {0} составляет {1} км/час, тип кузова: {2}", c3.ToString(), c3.RunMaxSpeed(), c3.RunCarType());

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AbstractFactory

{

abstract class CarFactory

{

public abstract AbstractCar CreateCar();

public abstract AbstractEngine CreateEngine();

public abstract AbstractType CreateType();

}

abstract class AbstractCar

{

public string Name { get; set; }

public abstract int MaxSpeed(AbstractEngine engine);

public abstract string CarType(AbstractType type);

}

abstract class AbstractEngine

{

public int max\_speed { get; set; }

}

abstract class AbstractType

{

public string Type { get; set; }

}

class FordFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new FordCar("Форд");

}

public override AbstractEngine CreateEngine()

{

return new FordEngine();

}

public override AbstractType CreateType()

{

return new FordType();

}

}

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 CarType(AbstractType type) { string tp = type.Type; return tp; }

public override string ToString()

{

return "Автомобиль " + Name;

}

}

class FordEngine : AbstractEngine

{

public FordEngine()

{

max\_speed = 220;

}

}

class FordType : AbstractType

{

public FordType()

{

Type = "Седан";

}

}

class AudiFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new AudiCar("Ауди");

}

public override AbstractEngine CreateEngine()

{

return new AudiEngine();

}

public override AbstractType CreateType()

{

return new AudiType();

}

}

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 CarType(AbstractType type) { string tp = type.Type; return tp; }

public override string ToString()

{

return "Автомобиль " + Name;

}

}

class AudiEngine : AbstractEngine

{

public AudiEngine()

{

max\_speed = 300;

}

}

class AudiType : AbstractType

{

public AudiType()

{

Type = "Хечбэк";

}

}

class Client

{

private AbstractCar abstractCar;

private AbstractEngine abstractEngine;

private AbstractType abstractType;

public Client(CarFactory car\_factory)

{

abstractCar = car\_factory.CreateCar();

abstractEngine = car\_factory.CreateEngine();

abstractType = car\_factory.CreateType();

}

public int RunMaxSpeed()

{

return abstractCar.MaxSpeed(abstractEngine);

}

public string RunCarType()

{

return abstractCar.CarType(abstractType);

}

public override string ToString()

{

return abstractCar.ToString();

}

}

class PegoFactory : CarFactory

{

PegoFactory() { }

static Lazy<PegoFactory> myPegoFactory = new Lazy<PegoFactory>(() => new PegoFactory());

public static PegoFactory MyPegoFactory

{

get { return myPegoFactory.Value; }

}

public override AbstractCar CreateCar()

{

return new PegoCar("Пежо");

}

public override AbstractEngine CreateEngine()

{

return new PegoEngine();

}

public override AbstractType CreateType()

{

return new PegoType();

}

}

class PegoCar : AbstractCar

{

public PegoCar(string name) { Name = name; }

public override int MaxSpeed(AbstractEngine engine) { int ms = engine.max\_speed; return ms; }

public override string CarType(AbstractType type) { string tp = type.Type; return tp; }

public override string ToString()

{

return "Автомобиль " + Name;

}

}

class PegoEngine : AbstractEngine

{

public PegoEngine()

{

max\_speed = 350;

}

}

class PegoType : AbstractType

{

public PegoType()

{

Type = "Кроссовер";

}

}

}

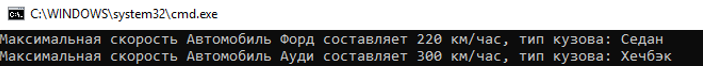
Контрольное задание

1. В разработанное приложение добавьте класс для новой конкретной фабрики, создающей новый автомобиль, например, Audi.

2. Добавьте в конфигурацию автомобиля новое свойство – тип кузова.

3. Проанализируйте трудоемкость вносимых изменений.

Смотреть выше. Добавила класс Audi и новое свойство – тип кузова.



Упражнение 3. Фабричный метод (Factory Method)

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

*Код упражнения:*

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 drTr = new TaxiTransCom("Пьяный водитель");

TransportService drinkService = drTr.Create("Пьяное такси", 5);

double dist1 = 20.0;

Print(drinkService, dist1);

}

private static void Print(TransportService compTax, double distg)

{

Console.WriteLine("Компания {0}, расстояние {1}, стоимость: {2}", compTax.ToString(), distg, compTax.CostTransportation(distg));

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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);

}

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);

}

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;

}

}

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;

}

}

class TaxiTransCom : TransportCompany

{

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

public override TransportService Create(string n, int c) { return new TaxiServices(Name, c); }

}

class ShipTransCom : TransportCompany

{

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

public override TransportService Create(string n, int t) { return new Shipping(Name, t); }

}

class DrinkTrans: TransportCompany

{

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

public override TransportService Create(string n, int c) { return new TaxiServices(Name, c); }

}

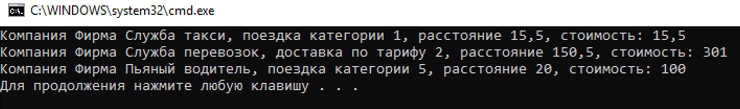
}

Контрольное задание

1. В разработанное приложение добавьте поддержку новой услуги, например, «пьяный водитель».

2. Проанализируйте трудоемкость вносимых изменений.

Добавил услугу «Пьяный водитель», для этого нужно было просто добавить новый класс.



Упражнение 4. Одиночка (Singleton)

В этом упражнении вы создадите 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 вы по шаблону «Одиночка» измените класс журнала, тем самым гарантируете, что у этого класса будет создаваться единственный экземпляр и что этот экземпляр будет легко доступен в любой точке приложения.

*Код упражнения:*

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);

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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)

{

Console.WriteLine(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("-------------------------------");

}

}

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;

}

}

}

Контрольное задание

В приложении AbstractFactory реализуйте класс конкретной фабрики с помощью паттерна «Одиночка».

*Код класса Pego с паттерном «Одиночка»:*

class PegoFactory : CarFactory

{

PegoFactory() { }

static Lazy<PegoFactory> myPegoFactory = new Lazy<PegoFactory>(() => new PegoFactory());

public static PegoFactory MyPegoFactory

{

get { return myPegoFactory.Value; }

}

public override AbstractCar CreateCar()

{

return new PegoCar("Пежо");

}

public override AbstractEngine CreateEngine()

{

return new PegoEngine();

}

public override AbstractType CreateType()

{

return new PegoType();

}

}

class PegoCar : AbstractCar

{

public PegoCar(string name) { Name = name; }

public override int MaxSpeed(AbstractEngine engine) { int ms = engine.max\_speed; return ms; }

public override string CarType(AbstractType type) { string tp = type.Type; return tp; }

public override string ToString()

{

return "Автомобиль " + Name;

}

}

class PegoEngine : AbstractEngine

{

public PegoEngine()

{

max\_speed = 350;

}

}

class PegoType : AbstractType

{

public PegoType()

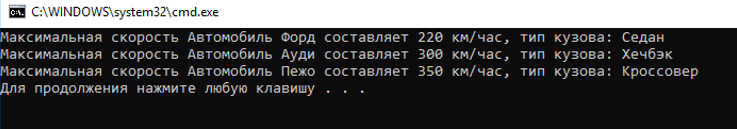
{

Type = "Кроссовер";

}

}

}



Упражнение 5. Стратегия (Strategy)

В этом упражнении вы создадите приложение, в котором реализуете на основе паттерна «Стратегия» работу 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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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 = { 0, 5, 10, 2, 4, 1, 14, 23, 126, 66 };

sort = new BubleSort();

context = new Context(sort, arr3);

context.Sort();

context.PrintArray();

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Strategy

{

abstract class StrategySort

{

public string Title { get; set; }

public abstract void Sort(int[] array);

}

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;

}

}

}

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;

}

}

}

}

class BubleSort : StrategySort

{

public BubleSort()

{

Title = "Сортировка пузырьком";

}

public override string ToString()

{

return Title;

}

public override void Sort(int[] array)

{

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

{

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

{

if (array[j] > array[j + 1])

{

int temp = array[j];

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

array[j + 1] = temp;

}

}

}

}

}

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++)

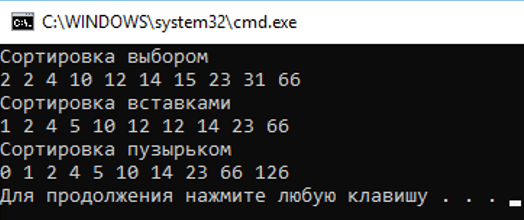
Console.Write(array[i] + " ");

Console.WriteLine();

}

}

}



Контрольное задание

Примените паттерн «Стратегия» для проектирования сложного алгоритма приложения навигатора. Оно должно иметь возможность показывать карту, реализовывать поиск и прокладку маршрута по автодорогам, пеших маршрутов, маршрутов по велодорожкам, на общественном транспорте, а также маршруты посещения достопримечательностей. Конкретно алгоритмы можно не реализовывать, только спроектировать общую структуру классов.

*Код задания:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Navigator1

{

class Program

{

static void Main(string[] args)

{

NavigatorActions n1 = new Map();

Context context = new Context(n1);

context.PrintAction();

context.Action();

n1 = new SearchCar();

context = new Context(n1);

context.PrintAction();

context.Action();

n1 = new SearchMan();

context = new Context(n1);

context.PrintAction();

context.Action();

n1 = new SearchByke();

context = new Context(n1);

context.PrintAction();

context.Action();

n1 = new SearchTransport();

context = new Context(n1);

context.PrintAction();

context.Action();

n1 = new SearchSights();

context = new Context(n1);

context.PrintAction();

context.Action();

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Navigator1

{

abstract class NavigatorActions

{

public string Title { get; set; }

public abstract void Action();

}

class Map : NavigatorActions

{

public Map()

{

Title = "Показать карту";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class SearchCar : NavigatorActions

{

public SearchCar()

{

Title = "Поиск и прокладка маршрута по автодорогам";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class SearchMan : NavigatorActions

{

public SearchMan()

{

Title = "Поиск и прокладка пешего маршрута";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class SearchByke : NavigatorActions

{

public SearchByke()

{

Title = "Поиск и прокладка маршрута по велодорожкам";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class SearchTransport : NavigatorActions

{

public SearchTransport()

{

Title = "Поиск и прокладка маршрута на общ.транспорте";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class SearchSights : NavigatorActions

{

public SearchSights()

{

Title = "Поиск и прокладка маршрута по достопримечательностям";

}

public override string ToString()

{

return Title;

}

public override void Action()

{

Console.WriteLine("Отработано");

}

}

class Context

{

NavigatorActions nav;

public Context(NavigatorActions nav)

{

this.nav = nav;

}

public void Action()

{

nav.Action();

}

public void PrintAction()

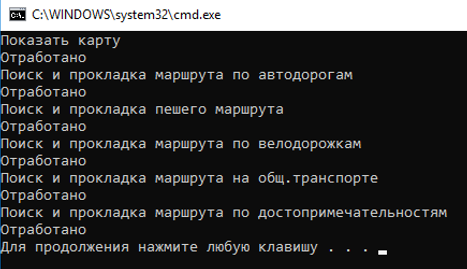
{

Console.WriteLine(nav.ToString());

}

}

}



Упражнение 6. Шаблонный метод (Template Method)

В этом упражнении вы создадите приложение, в котором реализуете 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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TemplateMethod

{

class Program

{

static void Main(string[] args)

{

Progression val = new ArithmeticProgression(1, 6, 3);

val.Progress();

Progression val1 = new GeometricProgression(1, 6, 3);

val.Progress();

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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();

}

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 fF = First;

do

{

progList.Add(fF);

fF = fF \* H;

}

while (fF < Last);

}

}

}

Упражнение 7. Фасад (Facade)

В этом упражнении вы реализуете 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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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();

var meal = new Cook(drive, power, notification);

meal.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());

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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 = "Поверот налево";

}

public void TurlRight()

{

Twist = "Поверот направо";

}

public void Stop()

{

Twist = "Стоп";

}

public override string ToString()

{

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

return s;

}

}

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;

}

}

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;

}

}

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();

}

}

class Cook

{

private Drive \_drive;

private Power \_power;

private Notification \_notification;

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

{

\_drive = drive;

\_power = power;

\_notification = notification;

}

public void Defrost()

{

Console.WriteLine("С какой мощностью готовить еду?");

\_power.MicrowavePower = int.Parse(Console.ReadLine());

\_notification.StartNotification();

\_drive.TurlRight();

\_drive.TurlLeft();

\_drive.TurlRight();

\_drive.TurlLeft();

\_drive.TurlRight();

\_drive.TurlLeft();

\_drive.TurlRight();

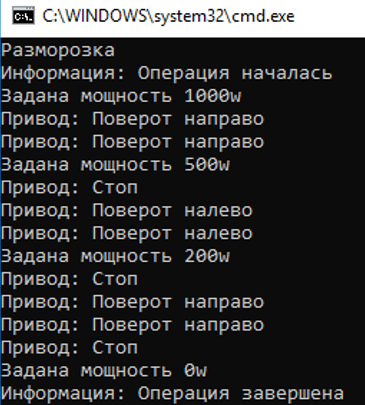
\_drive.TurlRight(); \_drive.Stop();

\_notification.StopNotification();

}

}

}



Контрольное задание

Добавьте в класс-фасад метод, реализующий приготовление продукта (алгоритм операций на ваше усмотрение). Протестируйте работу программы. Таким образом, реализован фасадный объект, обеспечивающий общий интерфейс работы с системой и выполняющий обязанность по взаимодействию с её компонентами.

Добавила класс Cook:

class Cook

{

private Drive \_drive;

private Power \_power;

private Notification \_notification;

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

{

\_drive = drive;

\_power = power;

\_notification = notification;

}

public void Defrost()

{

Console.WriteLine("С какой мощностью готовить еду?");

\_power.MicrowavePower = int.Parse(Console.ReadLine());

\_notification.StartNotification();

\_drive.TurlRight();

\_drive.TurlLeft();

\_drive.TurlRight();

\_drive.TurlLeft();

\_drive.TurlRight();

\_drive.TurlLeft();

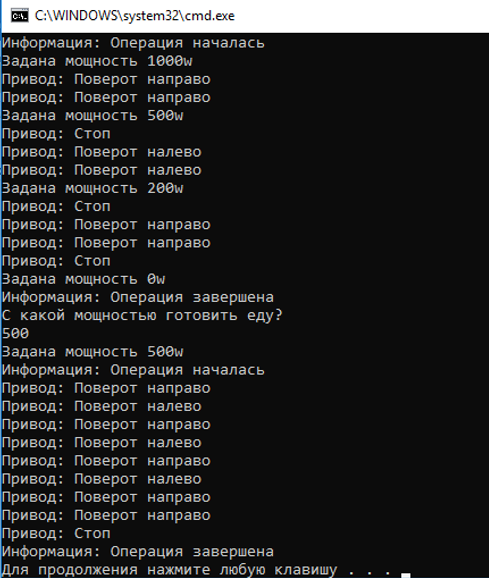
\_drive.TurlRight();

\_drive.TurlRight(); \_drive.Stop();

\_notification.StopNotification();

}

}



Упражнение 8. Цепочка обязанностей (Chain of Responsibility)

В этом упражнении вы реализуете систему отправки определенную сумму 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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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();

bankPaymentHandler.Successor = paypalPaymentHandler;

paypalPaymentHandler.Successor = moneyPaymentHnadler;

bankPaymentHandler.Handle(receiver);

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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 = ppt;

PayPalTransfer = mt;

}

}

abstract class PaymentHandler

{

public PaymentHandler Successor { get; set; }

public abstract void Handle(Receiver receiver);

}

}

*ConcretePaymentHandler.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ChainofResponsibility

{

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 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 PayPalPaymentHandler : PaymentHandler

{

public override void Handle(Receiver receiver)

{

if (receiver.PayPalTransfer == true)

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

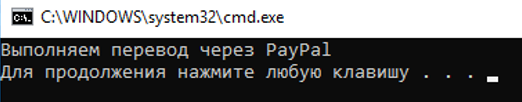
else if (Successor != null)

Successor.Handle(receiver);

}

}

}



Контрольное задание

1. Измените последовательность объектов-обработчиков цепочки и изучите результат.

2. Внесите изменение в значения передаваемых параметров при инициализации возможных используемых систем платежей. Протестируйте работу приложения.

При изменении последовательности вывода меняется надпись при запуске программы.

Упражнение 9. Команда (Command)

В этом упражнении вы реализуете 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 объекткоманда связан с этим устройством и умеет им управлять.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Command

{

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);

result = calculator.Subtraction(2);

Console.WriteLine(result);

result = calculator.Multiply(3);

Console.WriteLine(result);

result = calculator.Divide(10);

Console.WriteLine(result);

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Command

{

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 Subtraction : Command

{

public Subtraction(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 Multiply : Command

{

public Multiply(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 Divide : Command

{

public Divide(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 Subtraction(double operand)

{

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

}

public double Multiply(double operand)

{

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

}

public double Divide(double operand)

{

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

}

}

}

Контрольное задание

1. Добавьте три новые команды (вычитание, умножение и деление), для этого создайте соответствующие классы-наследники (удобно это сделать в файле ConcreteCommand.cs) от абстрактного класса Command и реализуйте его методы Execute() и UnExecute(), а также добавьте соответвующие методы в класс клиента – Calculator.

2. В классе ControlUnit добавьте поддержку многоуровневой отмены и повтора операций с помощью перегруженных версий методов Undo() и Redo().

Добавленные классы:  
class Subtraction : Command

{

public Subtraction(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 Multiply : Command

{

public Multiply(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 Divide : Command

{

public Divide(ArithmeticUnit unit, double operand)

{

this.unit = unit; this.operand = operand;

}

public override void Execute()

{

unit.Run('/', operand);

}

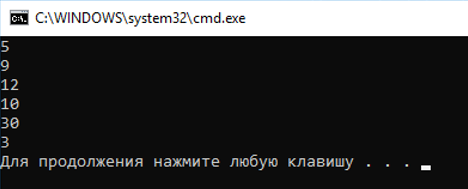
public override void UnExecute()

{

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

}

}



Упражнение 10. Декоратор (Decorator)

В этом упражнении вы реализуете 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.

*Код упражнения:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

//Добавила БМВ, люк и круизконтроль. Протестировала

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);

Renault bmw = new Renault("БМВ", "BMW i8 Roadster", 1000.0);

Print(reno);

AutoBase mybmw = new CruisControl(bmw, "Круизконтроль");

Print(mybmw);

AutoBase newmybmw = new Window(new CruisControl(bmw, "Скорость"), "Люк");

Print(newmybmw);

}

private static void Print(AutoBase av)

{

Console.WriteLine(av.ToString());

}

}

}

*Class.cs:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

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;

}

}

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 BMW : AutoBase

{

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

{

Name = name;

Description = info;

CostBase = costbase;

}

public override double GetCost()

{

return CostBase \* 2;

}

}

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 2.0; }

}

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;

}

}

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;

}

}

class CruisControl: DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Круизконтроль";

Description = p.Description + ". " + this.Title + ". Новейшая система круизконтроля(регулировка скорости)";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 50.00;

}

}

class Window : DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Люк";

Description = p.Description + ". " + this.Title + ". Открывающийся люк на крыше машины";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 10.00;

}

}

}

Контрольное задание

В разработанное приложение добавьте класс для нового автомобиля и две-три новые функциональные возможности.

Я добавил круизконтроль и люк.

*Добавленные классы:*

class CruisControl: DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Круизконтроль";

Description = p.Description + ". " + this.Title + ". Новейшая система круизконтроля(регулировка скорости)";

}

public override double GetCost()

{

return AutoProperty.GetCost() + 50.00;

}

}

class Window : DecoratorOptions

{

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

{

AutoProperty = p;

Name = p.Name + ". Люк";

Description = p.Description + ". " + this.Title + ". Открывающийся люк на крыше машины";

}

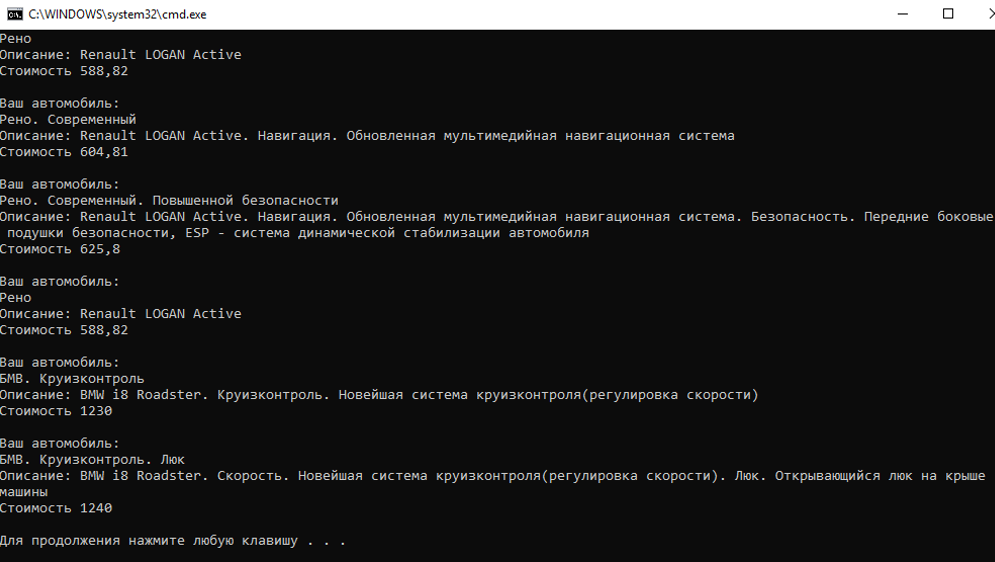
public override double GetCost()

{

return AutoProperty.GetCost() + 10.00;

}

}



**Вывод:**

В данной работе были изучены и применены на практике следующие шаблоны GoF: абстрактная фабрика, адаптер, фабричный метод, синглтон, стратегия, фасад, цепочка обязанностей, декоратор, команда и шаблонный метод. Использование данных паттернов позволит упростить проектирование, реализацию и поддержку разрабатываемой системы.