**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Факультет инфокоммуникационных технологий

Дисциплина: Проектирование инфокоммуникационных систем

Отчет к практической работе №4

**«Шаблоны проектирования Gang-of-Four»**

Выполнила:

студентка группы К4113с

Трифонова Альбина

Проверил:

ктн, доцент ИКТ,

Осипов Никита Алексеевич

Санкт-Петербург,

2020 г.

**Цель работы:** изучить шаблоны GoF, рассматриваемые в упражнениях.

***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.

Класс Fahrenheit реализует метод получения случайного значения температуры.

class Fahrenheit : FahrInterface

{

Random far;

public Fahrenheit()

{

far = new Random();

}

public double GetRandomFahr()

{

double grad = far.NextDouble() \* 10;

return grad;

}

}

Интерфейс, к которому имеет доступ датчик, выдающий данные в Фаренгейтах (измерение температуры моделируется получением случайного числа):

interface FahrInterface

{

double GetRandomFahr();

}

Адаптер:

class CelsAdapt

{

Fahrenheit f;

double f\_grad;

public CelsAdapt(Fahrenheit fr)

{

f = fr;

}

public double GetFahr()

{

f\_grad = f.GetRandomFahr();

return f\_grad;

}

public double FahrToCels()

{

double cels;

cels = (f\_grad - 32) \* 5 / 9;

return cels;

}

}

Класс Program реализует только вывод значений, переведенных в Цельсия.

class Program

{

static void Main(string[] args)

{

Fahrenheit fahrenheit = new Fahrenheit();

CelsAdapt celsAdapt = new CelsAdapt(fahrenheit);

Console.WriteLine("Получено град. в Фаренгейт: {0}, переведено в {1} град. Цельсия", celsAdapt.GetFahr(), celsAdapt.FahrToCels());

}

}

Реализуемая диаграмма классов:



***Упражнение 2. 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.

Контрольное задание.

1. Добавление параметра «Кузов»

namespace AbstractFactory

{

abstract class CarFactory

{

public abstract AbstractCar CreateCar();

public abstract AbstractEngine CreateEngine();

public abstract AbstractBody CreateBody();

}

}

namespace AbstractFactory

{

abstract class AbstractCar

{

public string Name { get; set; }

public abstract int MaxSpeed(AbstractEngine engine);

public abstract string Type(AbstractBody body);

}

1. Добавление класса абстрактного кузова

namespace AbstractFactory

{

class AbstractBody

{

public string body\_type { get; set; }

}

}

1. Реализация класса в конкретной марке

class AudiBody:AbstractBody

{

public AudiBody()

{

body\_type = "седан-баклажан";

}

}

class AudiFactory : CarFactory

{

public override AbstractCar CreateCar()

{

return new AudiCar("Ауди");

}

public override AbstractEngine CreateEngine()

{

return new AudiEngine();

}

public override AbstractBody CreateBody()

{

return new AudiBody();

}

}

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 Type(AbstractBody body)

{

string t = body.body\_type;

return t;

}

public override string ToString()

{

return "Машина " + Name;

}

}

Вывод в консоли:

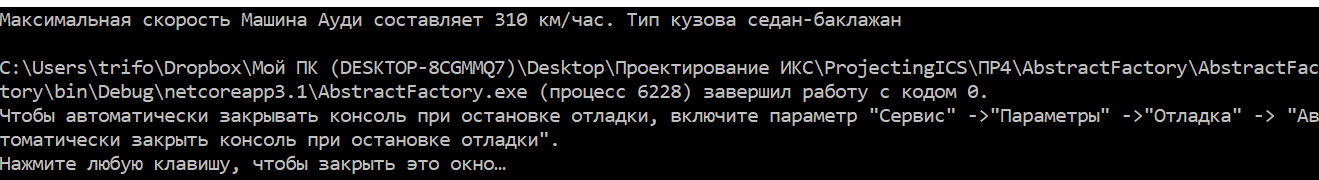


Диаграмма классов для паттерна Abstract Factory:



***Упражнение 3. 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.

Создание новой компании по реализации услуги пьяный водитель:

class DrunkDriverTransCom : TransportCompany

{

public DrunkDriverTransCom(string name) : base(name)

{ }

public override TransportService Create(string n, int c)

{

return new DrunkDriver(Name, c);

}

}

Создание услуги пьяный водитель:

class DrunkDriver : TransportService

{

public int CategoryOfCar { get; set; }

public DrunkDriver(string name, int category) : base(name)

{

/\*1 - Легковой автомобиль

2 - Грузовой автомобиль

3 - Минивен

4 - Легковой автомобиль (преиум класс)\*/

CategoryOfCar = category;

}

public override double CostTransportation(double distance)

{

switch (CategoryOfCar)

{

case 1:

return distance \* 1;

case 2:

return distance \* 3;

case 3:

return distance \* 1.5;

case 4:

return distance \* 5;

default:

return distance \* 1;

}

}

public override string ToString()

{

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

return s;

}

}

Class Program:

class Program

{

static void Main(string[] args)

{

TransportCompany trCom = new DrunkDriverTransCom("Служба 'Пьяный водитель'");

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

double dist = 15.5;

Print(compService, dist);

}

private static void Print(TransportService compTax, double distg)

{

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

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

}

}

Диаграмма классов:



***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.

Диаграмма классов:

******

***Упражнение 5. Стратегия (Strategy)***

Добавление нового класса Сортировки пузырьковым методом:

class BubbleSort : StrategySort

{

public BubbleSort()

{

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 = { 13, 10, 55, 0, 2, 23, 5, 89, 34, 11 };

sort = new BubbleSort();

context = new Context(sort, arr3);

context.Sort();

context.PrintArray();

}

}

Вывод:

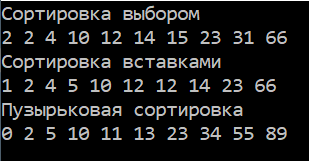


Диаграмма классов:



***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.

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);

}

}

Program:

static void Main(string[] args)

{

Console.Write("Введите первое число и последние, а также шаг соответственно (f, l, h): ");

int f = Convert.ToInt32(Console.ReadLine());

int l = Convert.ToInt32(Console.ReadLine());

int h = Convert.ToInt32(Console.ReadLine());

Progression val = new GeometricProgression(f, l, h);

val.TemplateMethod();

}

Вывод в консоли:

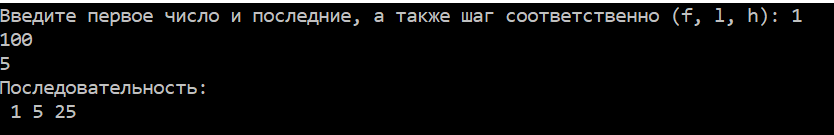


Диаграмма классов:



***Вывод:*** паттерны GoF предназначены для построения решений для тех или иных задач. Эти паттерны позволяют пользоваться уже готовыми эффективными решениями, не прибегая к поиску нового. Каждый паттерн имеет ряд достоинств и недостатков, которые стоит учитывать при выборе одного из них для решения своих задач.

На основании изученных паттернов были внесены изменения в диаграмму классов системы автоматизации работы администраторов в детской школе дополнительного образования.



Красным цветом отмечены связи, добавленные в диаграмму по паттернам Abstract Factory и Adapter.

*Abstract Factory*. Администратор обрабатывает журнал посещений и расписание занятий преподавателей. На основе абстрактной фабрики формирования расписания формируется расписание для каждого преподавателя, по которому преподаватель может отслеживать свои занятия и считать отработанные часы. SchedulePrep1 (фабрика) создает журнал, по которому можно отследить посещение занятий по каждому преподавателю.

*Adapter.* Для создания задачи необходимо понимать, какой статус у обрабатываемого клиента, чтобы установить тип задачи. Сам класс Task таким методом не обладает и не имеет доступа к классу Client, поэтому был создан класс Adapter, который позволяет преобразовать статус клиента в тип задачи.