using System;

using System.Collections;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Threading;

using System.Threading.Tasks;

using System.Xml.Serialization;

using System.IO;

namespace \_1\_2

{

abstract class Transport

{

public string name;

public Transport(string name)

{

this.name = name;

}

}

public interface IAirable

{

void Check();

void Fly();

}

public interface IAir

{

void Check();

}

class Air : Transport, IAirable, IAir

{

private int countOfPassengers = 0;

private int speed = 0;

private string status = null;

public Air(string name) : base(name)

{

}

public int CountOfPassengers

{

get { return countOfPassengers; }

set { countOfPassengers = value; }

}

public int Speed

{

get { return speed; }

set { speed = value; }

}

enum Status\_enum

{

Fly,

Ready,

Stop

}

public string Status

{

get { return status; }

set { status = value; }

}

void IAirable.Check()

{

if (countOfPassengers == 0 && speed == 0)

{

status = Status\_enum.Stop.ToString();

}

if (countOfPassengers > 0 && speed == 0)

{

status = Status\_enum.Ready.ToString();

}

if (countOfPassengers > 0 && speed > 0)

{

status = Status\_enum.Fly.ToString();

}

}

void IAir.Check()

{

if (countOfPassengers > 20 && countOfPassengers < 100)

{

Console.WriteLine("Ready");

}

}

public void Fly()

{

if (status == Status\_enum.Fly.ToString())

{

Console.WriteLine("Airplane is flying");

}

else

{

throw new Exception("Airplane is not flying");

}

}

}

class Program

{

static void Main(string[] args)

{

string way = @"C:\Users\dimat\Desktop\Экзамены\_3сем\ООП\Реализация задач\1\_2\1\_2\text.txt";

using (StreamWriter sw = new StreamWriter(way, false, System.Text.Encoding.Default))

{

Air air = new Air("Airplane");

air.CountOfPassengers = 10;

air.Speed = 0;

((IAirable)air).Check();

//((IAir)air).Check();

sw.WriteLine("Status: " + air.Status);

Console.WriteLine(air.Status);

air.Speed = 100;

((IAirable)air).Check();

sw.WriteLine("Status: " + air.Status);

((IAirable)air).Fly();

// Создать коллекцию из Air и добавить 5 объектов.

// С помощью linq запросов вывести количество самолетов, находящихся в Status = fly, а так же посчитать среднюю их скорость.

List<Air> airList = new List<Air>();

airList.Add(new Air("Airplane1"));

airList.Add(new Air("Airplane2"));

airList.Add(new Air("Airplane3"));

airList[0].CountOfPassengers = 10;

airList[0].Speed = 200;

((IAirable)airList[0]).Check();

airList[1].CountOfPassengers = 10;

airList[1].Speed = 400;

((IAirable)airList[1]).Check();

airList[2].CountOfPassengers = 10;

airList[2].Speed = 0;

((IAirable)airList[2]).Check();

var flyAir = airList.Where(x => x.Status == "Fly");

sw.WriteLine("Fly = " + flyAir.Count());

Console.WriteLine("Fly = " + flyAir.Count());

var averageSpeed = airList.Where(x => x.Status == "Fly").Average(x => x.Speed);

sw.WriteLine("Average speed = " + averageSpeed);

Console.WriteLine("Average speed = " + averageSpeed);

}

}

}

}

using System.Runtime.Serialization.Json;

using System.Text.Json;

using System.IO;

namespace \_2\_6

{

public enum Status

{

Signin,

Signout

}

class User

{

private string email;

private string password;

private Status status;

public override string ToString()

{

return "Email: " + email + " Password: " + password + " Status: " + status;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

User user = obj as User;

if (user as User == null)

return false;

return email == user.email && password == user.password && status == user.status;

}

public override int GetHashCode()

{

return email.GetHashCode() + password.GetHashCode() + status.GetHashCode();

}

// Перегрузить метод compareTo стандартного интерфейса IComparable, который сравнивает объекты по полю email

public int CompareTo(object obj)

{

if (obj == null)

return 1;

User user = obj as User;

if (user as User == null)

return 1;

return email.CompareTo(user.email);

}

public User(string email, string password, Status status)

{

this.email = email;

this.password = password;

this.status = status;

}

}

// Создайте класс WebNet, который содержит LinkedList<> всех пользователей и методами добавление и удаления.

class WebNet

{

public LinkedList<User> users = new LinkedList<User>();

public void AddUser(User user)

{

users.AddLast(user);

}

public void RemoveUser(User user)

{

users.Remove(user);

}

public void PrintUsers()

{

Console.WriteLine("Users:");

foreach (User user in users)

{

Console.WriteLine(user);

}

}

}

class Program

{

static void Main(string[] args)

{

User user1 = new User("dimatruba2004@yandex.ru", "123456", Status.Signin);

User user2 = new User("desrvdgf@mail.ru", "123456", Status.Signout);

User user3 = new User("dimatruba2004@yandex.ru", "123456345", Status.Signin);

Console.WriteLine(user1);

Console.WriteLine(user2);

Console.WriteLine(user3);

Console.WriteLine(user1.Equals(user2));

Console.WriteLine(user1.Equals(user3));

Console.WriteLine(user1.CompareTo(user3));

Console.WriteLine(user1.CompareTo(user2));

// Создайте объект github и добавьте в список всех пользователей

WebNet github = new WebNet();

github.AddUser(user1);

github.AddUser(user2);

github.AddUser(user3);

github.PrintUsers();

// Linq запрос для поиска пользователей с определенным статусом

var users = from user in github.users

where user.ToString().Contains("Signin")

select user;

Console.WriteLine("Users with status Signin:");

foreach (User user in users)

{

Console.WriteLine(user);

}

string json = JsonSerializer.Serialize(github);

Console.WriteLine(json);

File.WriteAllText("github.json", json);

}

}

}

namespace \_3\_2

{

class Location

{

public int lat { get; set; }

public int lon { get; set; }

public int speed { get; set; }

public Location(int lat, int lon, int speed)

{

this.lat = lat;

this.lon = lon;

this.speed = speed;

}

}

public enum Status

{

free,

busy

}

class Taxi

{

public string number { get; set; }

public Location location { get; set; }

public Status status;

public Taxi(string number, Location location, Status status)

{

this.number = number;

this.location = location;

this.status = status;

}

public override string ToString()

{

return "Taxi number: " + number + " Location: " + location.lat + " " + location.lon + " Speed: " + location.speed + " Status: " + status;

}

}

class Park<T>

{

public List<T> list = new List<T>();

public void Add(T item)

{

list.Add(item);

}

public void Remove(T item)

{

list.Remove(item);

}

public void Find(Predicate<T> predicate)

{

foreach (T item in list)

{

if (predicate(item))

{

Console.WriteLine(item);

}

}

}

public override string ToString()

{

string str = "";

foreach (T item in list)

{

str += item + "\n";

}

return str;

}

}

class Program

{

static void Main(string[] args)

{

// Объект uber - экземпляр класса Park, который содержит в себе коллекцию объектов класса Taxi

Park<Taxi> uber = new Park<Taxi>();

// Добавление объектов в коллекцию uber(4 объекта)

uber.Add(new Taxi("A123AA", new Location(10, 10, 10), Status.free));

uber.Add(new Taxi("B123BB", new Location(20, 20, 20), Status.free));

uber.Add(new Taxi("C123CC", new Location(30, 30, 30), Status.free));

uber.Add(new Taxi("D123DD", new Location(40, 40, 40), Status.free));

Console.WriteLine(uber.ToString());

// Отсортировать такси по расстоянию до пассажира, координаты пассажира ввести с клавиатуры

Console.WriteLine("Enter the coordinates of the passenger");

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

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

uber.list.Sort((x, y) => Math.Abs(x.location.lat - lat) + Math.Abs(x.location.lon - lon) - Math.Abs(y.location.lat - lat) - Math.Abs(y.location.lon - lon));

Console.WriteLine(uber.ToString());

// Найти ближайшее к пассажиру

var taxi = uber.list.OrderBy(x => Math.Abs(x.location.lat - lat) + Math.Abs(x.location.lon - lon)).First();

Console.WriteLine(taxi);

// Записать его в файл

File.WriteAllText(@"C:\Users\dimat\Desktop\Экзамены\_3сем\ООП\Реализация задач\3\_2\3\_2\Taxi.txt", taxi.ToString());

}

}

}

namespace \_3\_3

{

public class SomeString : IComparer<SomeString>

{

public string str;

public SomeString(string str)

{

this.str = str;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

SomeString str = obj as SomeString;

return this.str.Length == str.str.Length && this.str[0] == str.str[0] && this.str[this.str.Length - 1] == str.str[str.str.Length - 1];

}

public int Compare(SomeString x, SomeString y)

{

if (x.str.Length > y.str.Length)

return 1;

else if (x.str.Length < y.str.Length)

return -1;

else

return 0;

}

public static SomeString operator +(SomeString s1, char a1)

{

return new SomeString(s1.str + a1);

}

public static SomeString operator -(SomeString s1, char a1)

{

try

{

if (s1.str.Length == 0)

throw new Exception("String is empty");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

return new SomeString(s1.str = s1.str.Remove(0, 1));

}

}

public static class StringExtention

{

public static int Count(this SomeString s)

{

int count = 0;

foreach (var item in s.str)

{

if (item == ' ')

count++;

}

return count;

}

public static string Remove(this SomeString s)

{

foreach (var item in s.str)

{

if (item == '.' || item == ',' || item == '!' || item == ';' || item == '-')

{

s.str = s.str.Remove(item, ' ');

}

}

return s.str;

}

}

class Program

{

static void Main(string[] args)

{

// Заменить консольный вывод в файл

string path = @"C:\Users\dimat\Desktop\Экзамены\_3сем\ООП\Реализация задач\3\_3\3\_3\text.txt";

using (StreamWriter stream = new StreamWriter(path, false, System.Text.Encoding.Default))

{

SomeString s1 = new SomeString("qw.erty");

SomeString s2 = new SomeString("qw ert y");

stream.WriteLine(s1.Compare(s1, s2));

s1 = s1 + 'b';

s2 = s2 - ' ';

stream.WriteLine(s1.str);

stream.WriteLine(s2.str);

stream.WriteLine(StringExtention.Count(s1));

stream.WriteLine(StringExtention.Remove(s2));

SomeString[] somes = new SomeString[2];

somes[0] = s1;

somes[1] = s2;

var select = from s in somes

where s.Count() > 0

select s;

int sum = 0;

foreach (var item in select)

{

sum += item.Count();

}

stream.WriteLine(sum);

}

}

}

}

using System.IO;

using System.Text.Json;

namespace \_4\_5

{

interface INumber

{

int Number { get; set; }

}

class Bill : INumber

{

private int number;

public int Number

{

get { return number; }

set

{

if (value == 5 || value == 10 || value == 20 || value == 50 || value == 100)

{

number = value;

}

else

{

throw new Exception("Неверное значение");

}

}

}

}

class Wallet<T> where T : INumber

{

public List<T> bills = new List<T>();

public void AddBill(T bill)

{

if (bills.Sum(x => x.Number) + bill.Number > 200)

{

throw new Exception("Сумма купюр больше 200");

}

else

{

bills.Add(bill);

}

}

public void RemoveBill()

{

if (bills.Count == 0)

{

throw new Exception("Купюр нет");

}

else

{

bills.Remove(bills.OrderBy(x => x.Number).First());

}

}

}

class Program

{

static void Main(string[] args)

{

Wallet<Bill> wallet = new Wallet<Bill>();

wallet.AddBill(new Bill { Number = 5 });

wallet.AddBill(new Bill { Number = 10 });

wallet.AddBill(new Bill { Number = 20 });

wallet.AddBill(new Bill { Number = 50 });

wallet.AddBill(new Bill { Number = 50 });

wallet.RemoveBill();

// На основе LINQ посчитайте кол-во купюр каждого достоинства в кошельке.

var query = from bill in wallet.bills

group bill by bill.Number into g

select new { Number = g.Key, Count = g.Count() };

foreach (var item in query)

{

Console.WriteLine("Номинал {0} - {1} шт.", item.Number, item.Count);

}

// Сохраните данные в файл в формате JSON.

string json = JsonSerializer.Serialize(wallet.bills);

File.WriteAllText("wallet.json", json);

}

}

}

namespace \_7\_3

{

public class Button : CheckButton

{

public string caption;

(int x, int y) startpoint;

public int X

{

get

{

return startpoint.x;

}

set

{

value = startpoint.x;

}

}

public int Y

{

get

{

return startpoint.y;

}

set

{

value = startpoint.y;

}

}

public int w;

public int h;

public Button(string caption, int x, int y, int w, int h, State state)

{

this.caption = caption;

this.startpoint.x = x;

this.startpoint.y = y;

this.w = w;

this.h = h;

this.state = state;

}

public override string ToString()

{

return $"Caption: {caption}, X: {X}, Y: {Y}, W: {w}, H: {h}, State: {state}";

}

public override bool Equals(object? obj)

{

if (obj == null)

{

return false;

}

if (obj.GetType() != this.GetType())

{

return false;

}

Button b = (Button)obj;

return (b.caption == this.caption && b.w == this.w && b.h == this.h);

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public void Check()

{

if (state == State.check)

{

state = State.uncheck;

}

else

{

state = State.check;

}

}

public void Zoom(int a)

{

this.w = this.w \* a;

this.h = this.h \* a;

}

public int square;

public int Square()

{

square = w \* h;

return square;

}

}

public class CheckButton

{

public enum State

{

check = 1,

uncheck

}

public State state;

}

public delegate void click();

public delegate void zoom(int x);

public class User

{

public event click Click;

public event zoom Zoom;

public void Events()

{

Click.Invoke();

Zoom.Invoke(100);

}

}

class ButtonCollection

{

public LinkedList<Button> myList = new LinkedList<Button>();

public void Add(Button myButton)

{

myList.AddLast(myButton);

}

// Запрос поиска кнопок заданной площади

public void Search(double square)

{

var query = from Button b in myList

where b.Square() == square

select b;

foreach (var item in query)

{

Console.WriteLine(item);

}

}

}

class Program

{

static void Main(string[] args)

{

Button button1 = new Button("Кнопка1", 10, 45, 10, 21, CheckButton.State.check);

Console.WriteLine(button1);

Button button2 = new Button("Кнопка2", 5, 100, 71, 11, CheckButton.State.check);

Button button3 = new Button("Кнопка3", 75, 43, 10, 1, CheckButton.State.check);

Button button4 = new Button("Кнопка4", 25, 80, 97, 36, CheckButton.State.uncheck);

Button button5 = new Button("Кнопка5", 50, 10, 42, 12, CheckButton.State.check);

Button button6 = new Button("Кнопка5", 50, 10, 42, 1, CheckButton.State.check);

Console.WriteLine(button2.Equals(button4));

Console.WriteLine(button5.Equals(button6));

Console.WriteLine(button3);

button3.Zoom(2);

Console.WriteLine(button3);

Console.WriteLine(button4);

User user = new User();

user.Click += button1.Check;

user.Click += button2.Check;

user.Click += button3.Check;

user.Zoom += button4.Zoom;

user.Zoom += button5.Zoom;

user.Zoom += button6.Zoom;

user.Events();

Console.WriteLine(button4);

ButtonCollection buttonCollection = new ButtonCollection();

buttonCollection.Add(button1);

buttonCollection.Add(button2);

buttonCollection.Add(button3);

buttonCollection.Add(button4);

buttonCollection.Add(button5);

buttonCollection.Add(button6);

Console.WriteLine("\n\n\n");

// Поиск кнопок с заданной площадью

buttonCollection.Search(420000);

// Запрос поиска кнопок типа CheckButton

var query = from Button b in buttonCollection.myList

where b.GetType() == typeof(CheckButton)

select b;

foreach (var item in query)

{

Console.WriteLine(item);

}

}

}

}

namespace \_8\_3

{

public interface IManage

{

float MaxAvg();

}

public enum Form

{

our = 1,

your,

my

}

public class ZiroException : Exception

{

public ZiroException(string message) : base(message)

{

Console.WriteLine(message);

}

}

public class Company : IManage

{

public string name { get; set; }

public int count { get; set; }

Form form { get; set; }

public int year1 { get; set; }

public int year2 { get; set; }

public int year3 { get; set; }

public int year4 { get; set; }

public Company(string \_name, int \_count, Form \_form, int \_year1, int \_year2, int \_year3, int \_year4)

{

this.name = \_name;

this.count = \_count;

this.form = \_form;

this.year1 = \_year1;

this.year2 = \_year2;

this.year3 = \_year3;

this.year4 = \_year4;

}

public override string ToString()

{

return $"{name} {count} {form} {year1} {year2} {year3} {year4}";

}

public (int, int) MinMaxMoney()

{

List<int> money = new List<int>();

money.Add(year1);

money.Add(year2);

money.Add(year3);

money.Add(year4);

int min = money.Min();

int max = money.Max();

var result = (min, max);

return result;

}

float IManage.MaxAvg()

{

float sum = 0;

float result;

sum = (float)(year1 + year2 + year3 + year4);

result = sum / 4;

return result;

}

public static Company operator ++(Company obj)

{

obj.count++;

return obj;

}

public static Company operator --(Company obj)

{

try

{

if (obj.count == 0)

throw new ZiroException("Null");

}

catch (ZiroException ex)

{

Console.WriteLine(ex.Message);

}

obj.count--;

return obj;

}

public static Company operator +(Company obj, int i)

{

obj.count = obj.count + i;

return obj;

}

}

public static class Extension

{

public static Company DeleteInfo(Company company)

{

company.year1 = 0;

company.year2 = 0;

company.year3 = 0;

company.year4 = 0;

return company;

}

}

class Program

{

static void Main(string[] args)

{

Company company = new Company("EPAM", 450, Form.your, 45, 57, 38, 39);

Console.WriteLine(company.MinMaxMoney());

Console.WriteLine(((IManage)company).MaxAvg());

Console.WriteLine(company.ToString());

company++;

Console.WriteLine(company.ToString());

company--;

Extension.DeleteInfo(company);

Console.WriteLine(company.ToString());

}

}

}

namespace \_8\_4

{

public interface IScore

{

int Amount { get; set; }

int AddMoney();

int RemMoney();

}

abstract class Human

{

DateTime Date { get; set; }

}

class Person : Human, IScore

{

public static int countobj;

static Person()

{

countobj = 0;

}

public string Name;

public string SecName;

public DateTime Date;

public int amount;

public int Amount

{

set

{

amount = value;

}

get

{

return amount;

}

}

public Person(string name, string secName, DateTime date, int amount)

{

Name = name;

SecName = secName;

Date = date;

Amount = amount;

countobj++;

}

public int AddMoney()

{

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

Amount = Amount + count;

return Amount;

}

public int RemMoney()

{

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

Amount = Amount - count;

return Amount;

}

public static void CountobjToString()

{

Console.WriteLine("Создано {0} объектов Person", countobj);

}

public override bool Equals(object obj)

{

Person a = obj as Person;

if (a.Date == this.Date)

{

return true;

}

else { return false; }

}

public override string ToString()

{

return ("Имя " + Name + " Фамилия " + SecName + " количество " + Amount + " дата " + Date);

}

}

class Bank : List<Person>

{

public void show()

{

Bank central = new Bank();

foreach (Person item in central)

{

Console.WriteLine(item.Name);

Console.WriteLine(item.SecName);

Console.WriteLine(item.Amount);

}

}

}

class Program

{

static void Main(string[] args)

{

DateTime time1 = new DateTime(2001, 5, 20);

DateTime time2 = new DateTime(2002, 12, 21);

DateTime time3 = new DateTime(2003, 6, 24);

DateTime time4 = new DateTime(2004, 7, 22);

Person person1 = new Person("Arsenii", "Mingazov", time1, 200);

Person person2 = new Person("Dima", "Radovid", time2, 200);

Person person3 = new Person("Jorj", "Geraklit", time3, 200);

Person person4 = new Person("Salam", "Abdul", time4, 200);

Console.WriteLine(person1.ToString());

Console.WriteLine(person1.AddMoney());

Console.WriteLine(person1.ToString());

Console.WriteLine(person1.RemMoney());

Console.WriteLine(person1.ToString());

Person.CountobjToString();

Console.WriteLine(person1.Equals(person2));

Console.WriteLine(person1.Equals(person3));

Console.WriteLine(person1.Equals(person4));

Console.WriteLine(person1.Equals(person1));

Bank belarus = new Bank();

belarus.Add(person1);

belarus.Add(person2);

belarus.Add(person3);

Bank alfa = new Bank();

alfa.Add(person4);

alfa.Add(person2);

alfa.Add(person3);

Bank central = new Bank();

central.Add(person1);

central.Add(person3);

central.show();

// Создать task для поиска клиента в банке по дате рождения

Task task = new Task(() =>

{

Console.WriteLine("Введите дату рождения");

DateTime date = Convert.ToDateTime(Console.ReadLine());

foreach (Person item in belarus)

{

if (item.Date == date)

{

Console.WriteLine(item.Name);

Console.WriteLine(item.SecName);

Console.WriteLine(item.Amount);

}

}

});

task.Start();

task.Wait();

}

}

}

using System.Collections;

namespace \_8\_7

{

public class AirPort

{

public AirPort()

{

airs = new List<Air>();

}

public List<Air> airs;

public void Add(Air obj)

{

airs.Add(obj);

}

public void Remove(Air obj)

{

airs.Remove(obj);

}

public void Pilot(AirPort obj)

{

var select = from o in airs

orderby o.time

select o;

foreach (var o in select)

{

Console.WriteLine(o);

}

}

}

public static class AirPortExtention

{

public static void Sort(this AirPort obj)

{

var selectbynumders = from t in obj.airs

where t.pilot.number >= 100

select t.pilot.number;

foreach (var t in selectbynumders)

{

Console.WriteLine(t);

}

}

}

public class Pilot

{

public string name;

public int number;

public Pilot(string name, int number)

{

this.name = name;

this.number = number;

}

}

public class Air : IComparable, IComparer<Air>

{

public string model { get; set; }

public Pilot pilot { get; set; }

public string napr { get; set; }

public string time { get; set; }

public Air(string model, Pilot pilot, string napr, string time)

{

this.model = model;

this.pilot = pilot;

this.napr = napr;

this.time = time;

}

public override string ToString()

{

return base.ToString() + " " + model + " " + pilot + " " + napr + " " + time;

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public int Compare(Air air1, Air air2)

{

if (air1.pilot.name.Length < air2.pilot.name.Length)

return -1;

else if (air1.pilot.name.Length > air2.pilot.name.Length)

return 1;

else

return 0;

}

public int CompareTo(object o)

{

Air air = o as Air;

if (air != null)

return this.time.CompareTo(air.time);

else

throw new Exception("Object is not a Air");

}

}

class Program

{

static void Main(string[] args)

{

Pilot pilot1 = new Pilot("Anna", 129);

Pilot pilot2 = new Pilot("Vlad", 97);

Air air1 = new Air("vupsen", pilot1, "Москва", "12:15");

Air air2 = new Air("pupsen", pilot2, "Санкт-Петербург", "12:14");

Console.WriteLine(air1.CompareTo(air2));

AirPort airport = new AirPort();

airport.Add(air1);

airport.Add(air2);

airport.Sort();

airport.Pilot(airport);

}

}

}

namespace Exam

{

interface IAction<T>

{

void Add(T obj);

void Remove(T obj);

void Clean();

void Info();

}

class NullSizeCollection : System.Exception

{

public NullSizeCollection(string message) : base(message)

{

Console.WriteLine("Коллекция пуста");

}

}

public class ExamCard<T> : IAction<T> where T : Student

{

List<T> list = new List<T>();

public List<T> List

{

get { return list; }

set { list = value; }

}

public void Add(T obj)

{

list.Add(obj);

}

public void Remove(T obj)

{

try

{

if (list.Count == 0)

{

throw new NullSizeCollection("Коллекция пустая");

}

else list.Remove(obj);

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

public void Clean()

{

try

{

if (list.Count == 0)

{

throw new NullSizeCollection("Коллекция пустая");

}

else list.Clear();

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

public void Info()

{

foreach (var item in list)

{

Console.WriteLine(item);

}

}

}

public class Student

{

public string Name;

public string Subject;

public int Mark;

public Student(string name, string subject, int mark)

{

this.Name = name;

this.Subject = subject;

this.Mark = mark;

}

public override string ToString()

{

return $"Name: {Name} Subject: {Subject} Mark: {Mark}";

}

}

class Program

{

static void Main(string[] args)

{

Student student1 = new Student("Ivan", "Math", 5);

Student student2 = new Student("Petr", "Physics", 3);

Student student3 = new Student("Sidor", "OAP", 7);

ExamCard<Student> examCard = new ExamCard<Student>();

IAction<Student> action = examCard;

action.Add(student1);

action.Add(student2);

action.Add(student3);

action.Info();

var query\_4 = from student in examCard.List

where student.Mark >= 4

select student;

Console.WriteLine("Students with mark >= 4");

Console.WriteLine(query\_4.Count());

foreach (var item in query\_4)

{

Console.WriteLine(item);

}

double average = 0;

int count = 0;

foreach (var item in query\_4)

{

average += item.Mark;

count++;

}

average = average / count;

Console.WriteLine("Average mark: " + average);

Random random = new Random();

foreach (var item in query\_4)

{

item.Mark += random.Next(1, 3);

Console.WriteLine(item);

}

}

}

}

namespace \_6\_1

{

class PinErrorException : Exception

{

public PinErrorException(string message) : base(message)

{

Console.WriteLine("Пароль введён неверно");

}

}

interface ICreditCard

{

void Add(int obj);

void Get(int obj);

}

public class CreditCard : ICreditCard

{

public int balance;

public int number;

private readonly int pin;

private readonly int pin2;

public CreditCard(int balance, int number, int pin, int pin2)

{

this.balance = balance;

this.number = number;

this.pin = pin;

this.pin2 = pin2;

}

public override string ToString()

{

return $"Balance - {balance}, Number - {number}";

}

public void CheckBalance()

{

int pinInput = 0;

while (true)

{

if (pinInput < 3)

{

try

{

Console.WriteLine("Введите pin: ");

int pin\_1 = Convert.ToInt32(Console.ReadLine());

if (pin\_1 == pin)

{

Console.WriteLine("Баланс: " + balance);

break;

}

else

{

pinInput++;

throw new PinErrorException("Пароль введён неверно");

}

}

catch (Exception ex)

{

Console.WriteLine();

}

}

else

{

try

{

Console.WriteLine("Введите pin2: ");

int pin\_2 = Convert.ToInt32(Console.ReadLine());

if (pin\_2 == pin2)

{

Console.WriteLine("Баланс: " + balance);

break;

}

else

{

throw new PinErrorException("Пароль введён неверно");

}

}

catch (PinErrorException ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

public void Add(int obj)

{

balance = this.balance + obj;

Console.WriteLine("Баланс пополнен на " + obj + " рублей и равен " + balance);

}

public void Get(int obj)

{

try

{

if (balance - obj >=0)

{

balance = this.balance - obj;

Console.WriteLine("Снято " + obj + " рублей. Баланс равен " + balance);

}

else

{

throw new Exception("Недостаточно средств(снимаете больше, чем на карте)");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

class Program

{

static void Main(string[] args)

{

CreditCard creditCard = new CreditCard(250, 19, 123, 321);

CreditCard creditCard2 = new CreditCard(150, 29, 1441, 1001);

CreditCard creditCard3 = new CreditCard(130, 39, 5555, 2004);

creditCard.CheckBalance();

((ICreditCard)creditCard).Add(45);

((ICreditCard)creditCard).Get(100);

List<CreditCard> creditCards = new List<CreditCard>();

creditCards.Add(creditCard);

creditCards.Add(creditCard2);

creditCards.Add(creditCard3);

var selectbymoney = from s in creditCards

where s.balance > 100 && s.balance < 200 && s.number.ToString().Contains("9")

select s.balance;

int sum = 0;

foreach (var item in selectbymoney)

{

sum += item;

Console.WriteLine(item);

}

Console.WriteLine("Сумма балансов: " + sum);

}

}

}

using System.Collections;

namespace \_8\_1

{

public interface IEnumerable

{

}

public class Item

{

public string name { get; set; }

public int ID { get; set; }

public double price { get; set; }

public Item(string name, int ID, int price)

{

this.name = name;

this.ID = ID;

this.price = price;

}

public override string ToString()

{

return $"Name: {name}, ID: {ID}, Price: {price}";

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public void OnSale()

{

this.price = this.price - (this.price \* 0.1);

Console.WriteLine($"sale is now");

}

}

public class Manager

{

public event \_Sale sale;

public void Sale()

{

if (sale != null)

sale();

}

}

public delegate void \_Sale();

public class Shop : IEnumerable

{

Queue<Item> queue = new Queue<Item>();

public void Add(Item obj)

{

queue.Enqueue(obj);

}

public void Remove(Item obj)

{

queue.Dequeue();

}

public void Delete(Item obj)

{

queue.TrimExcess();

}

}

class Program

{

static void Main(string[] args)

{

Item item1 = new Item("shirt", 1236, 2000);

Item item2 = new Item("dress", 3466, 1500);

Item item3 = new Item("boots", 4578, 3000);

Item item4 = new Item("shirt", 145, 3000);

Item item5 = new Item("shirt", 126, 5000);

Queue<Item> queue = new Queue<Item>();

queue.Enqueue(item1);

queue.Enqueue(item2);

queue.Enqueue(item3);

queue.Enqueue(item4);

queue.Enqueue(item5);

Console.WriteLine(item1.ToString());

Console.WriteLine(item2.GetHashCode());

foreach (Item a in queue)

{

Console.WriteLine(a);

}

Manager manager = new Manager();

manager.sale += item1.OnSale;

manager.sale += item3.OnSale;

manager.Sale();

foreach (Item a in queue)

Console.WriteLine(a);

Console.WriteLine();

// LINQ запрос, чтобы найти сумму товаров с заданным именем

var query = from Item in queue

where Item.name == "shirt"

select Item.price;

Console.WriteLine("Sum of prices: " + query.Sum());

}

}

}

using System.Collections;

namespace bstustud

{

public interface IClearnable

{

void Clearn();

}

public enum Specialization

{

POIT,

ISIT,

POIBMS,

DEIVI

}

class BSTUStudent

{

public string name;

public int group;

public int course;

public Specialization specialization;

public int mark1, mark2, mark3, mark4;

public BSTUStudent(string name, int group, int course, Specialization specialization, int mark1, int mark2, int mark3, int mark4)

{

this.name = name;

this.group = group;

this.course = course;

this.specialization = specialization;

this.mark1 = mark1;

this.mark2 = mark2;

this.mark3 = mark3;

this.mark4 = mark4;

}

public static (int min, int max, int avr) Getmarks(BSTUStudent obj)

{

var result = (min: 0, max: 0, avr: 0);

int[] nums = new int[4];

nums[0] = obj.mark1;

nums[1] = obj.mark2;

nums[2] = obj.mark3;

nums[3] = obj.mark4;

result.max = nums.Max();

result.min = nums.Min();

result.avr = (int)nums.Average();

return result;

}

public override string ToString()

{

return ("Имя " + name + " Группа " + group + " Курс " + course + " Специальность " + specialization + " Оценки " + mark1 + " " + mark2 + " " + mark3 + " " + mark4);

}

}

class SGroup : IClearnable

{

ArrayList list = new ArrayList();

public ArrayList GetList

{

get

{

return list;

}

}

public void Add(object obj)

{

list.Add(obj);

}

public void Remove(object obj)

{

list.Remove(obj);

}

public void Clearn()

{

list.Clear();

}

}

class Program

{

static void Main(string[] args)

{

BSTUStudent student1 = new BSTUStudent("Dmitry", 2, 5, Specialization.POIT, 6, 7, 8, 5);

BSTUStudent student2 = new BSTUStudent("Andrey", 1, 3, Specialization.ISIT, 4, 5, 8, 8);

BSTUStudent student3 = new BSTUStudent("Dasha", 3, 8, Specialization.POIBMS, 4, 4, 4, 9);

BSTUStudent student4 = new BSTUStudent("Shyra", 4, 10, Specialization.DEIVI, 7, 7, 7, 7);

var tuple = BSTUStudent.Getmarks(student1);

Console.WriteLine(tuple);

// Поиск через LINQ двух студентов с наибольшим средним баллом

BSTUStudent[] students = new BSTUStudent[4];

students[0] = student1;

students[1] = student2;

students[2] = student3;

students[3] = student4;

var result = (from student in students

orderby student.mark1 + student.mark2 + student.mark3 + student.mark4 descending

select student).Take(2);

foreach (var item in result)

{

Console.WriteLine(item);

}

SGroup listic = new SGroup();

listic.Add(student1);

listic.Add(student2);

listic.Add(student3);

listic.Add(student4);

foreach (BSTUStudent stud in listic.GetList)

{

Console.WriteLine(stud.name);

Console.WriteLine(stud.group);

Console.WriteLine(stud.specialization);

}

listic.Clearn();

Console.WriteLine("Список очищен");

foreach (BSTUStudent stud in listic.GetList)

{

Console.WriteLine(stud.name);

Console.WriteLine(stud.group);

Console.WriteLine(stud.specialization);

}

}

}

}

namespace \_8\_2

{

abstract class Function

{

public int X { get; set; }

public virtual double Func(double x, double c, double a, double b)

{

return a \* x \* x + b \* x + c;

}

public virtual double Func(double x, double a, double b)

{

return a \* x + b;

}

}

class Liner : Function

{

public double A { get; set; }

public double B { get; set; }

}

class Sqr : Function

{

public int C { get; set; }

}

public delegate string MyDelegate();

class ArrayFunc<T>

{

public static string ToString()

{

return "Work";

}

new public static string GetHashCode()

{

return "Work";

}

public static Func<int, int>[] funcs = new Func<int, int>[3];

}

// Индексатор

class ArrayIndex<T>

{

ArrayFunc<double>[] data;

public ArrayIndex()

{

data = new ArrayFunc<double>[5];

}

public ArrayFunc<double> this[int index]

{

get

{

return data[index];

}

set

{

data[index] = value;

}

}

}

class Program

{

static void Main()

{

MyDelegate[] delegates = new MyDelegate[2];

delegates[0] = ArrayFunc<string>.ToString;

delegates[1] = ArrayFunc<string>.GetHashCode;

ArrayFunc<string>.funcs[0] = i => i + 1;

ArrayFunc<string>.funcs[1] = i => i \* 2;

ArrayFunc<string>.funcs[2] = i => i - 3;

foreach (Func<int, int> func in ArrayFunc<string>.funcs)

{

Console.WriteLine(func(2));

}

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Xml;

using System.Xml.Linq;

using System.Xml.Serialization;

using System.Text.Json;

using System.Threading.Tasks;

namespace \_1\_1

{

class Program

{

static async Task Main(string[] args)

{

List<Rectangle> listochek = new List<Rectangle>();

Rectangle rec1 = new Rectangle(22, 24, 22, 25, "Синий");

Rectangle rec2 = new Rectangle(12, 14, 12, 15, "Красный");

Rectangle rec3 = new Rectangle(12, 14, 12, 15, "Черный");

Rectangle rec4 = new Rectangle(12, 14, 1, 1, "Белый");

Rectangle rec5 = new Rectangle(12, 14, 12, 15, "Оранжевый");

Rectangle rec6 = new Rectangle(12, 14, 12, 15, "Розовый");

listochek.Add(rec1);

listochek.Add(rec2);

listochek.Add(rec3);

listochek.Add(rec4);

listochek.Add(rec5);

listochek.Add(rec6);

rec2 = rec2 + 20;

Console.WriteLine(rec2.H + rec2.L);

var ordered = from i in listochek

orderby i.X

orderby i.Y

select i;

foreach (var i in ordered)

{

Console.WriteLine(i);

}

var linq = listochek.OrderBy(t => t.H).OrderBy(t => t.L).Last();

Console.WriteLine(linq);

Console.WriteLine(linq.ToString());

using (FileStream fs = new FileStream("user.json", FileMode.OpenOrCreate))

{

await JsonSerializer.SerializeAsync<Rectangle>(fs, rec1);

Console.WriteLine("Data has been saved to file");

}

}

}

interface Figure

{

void Print();

}

[Serializable]

class Rectangle : Figure

{

public int X { get; set; }

public int Y { get; set; }

public int H { get; set; }

public int L { get; set; }

public string Color { get; set; }

public Rectangle() { }

public Rectangle(int x, int y, string color)

{

X = x;

Y = y;

Color = color;

}

public Rectangle(int x, int y, int l, int h, string color) : this(x, y, color)

{

L = l;

H = h;

}

public static Rectangle operator +(Rectangle A, int i)

{

A.H += i;

A.L += i;

return A;

}

public int Sqr(Rectangle pl)

{

return pl.H \* pl.L;

}

public void Print()

{

Console.WriteLine($"Координаты: {X}, {Y}, длина: {L}, высота: {H}, цвет: {Color}");

}

public override string ToString()

{

return X + " " + Y + " " + L + " " + H + " " + Color;

}

}

}

using System.IO;

namespace \_5\_2

{

class OwnNotWorking : Exception

{

public OwnNotWorking(string message) : base(message)

{

Console.WriteLine("OwnNotWorking");

}

}

interface ICookable

{

void Cook();

void Check();

}

abstract class Device

{

int number { get; set; }

string name { get; set; }

}

class Own : Device, ICookable

{

public int Temp { get; set; }

public int Time { get; set; }

public Status status;

public enum Status

{

ready = 1,

cooking,

finish

}

public Own()

{

}

void ICookable.Cook()

{

string way = @"C:\Users\dimat\Desktop\Экзамены\_3сем\ООП\Реализация задач\5\_2\5\_2\text.txt";

using (StreamWriter sw = new StreamWriter(way, false, System.Text.Encoding.Default))

{

if (status == Status.cooking)

sw.WriteLine("Cooking");

else if (status == Status.ready)

sw.WriteLine("Ready");

else

sw.WriteLine("Finish");

}

}

void ICookable.Check()

{

if (Temp == 0 && Time > 0)

this.status = Status.ready;

if (Temp > 0 && Time > 0)

this.status = Status.cooking;

if (Temp > 0 && Time == 0)

this.status = Status.finish;

}

public override string ToString()

{

return $" {Time} {Temp} {status} ";

}

}

class Program

{

static void Main(string[] args)

{

List<Own> owns = new List<Own>();

Own own1 = new Own();

Own own2 = new Own();

own1.Temp = 70;

own1.Time = 15;

own1.status = Own.Status.cooking;

own2.Temp = 60;

own2.Time = 0;

own2.status = Own.Status.cooking;

owns.Add(own1);

owns.Add(own2);

var select = from i in owns

where i.status == Own.Status.cooking

select i;

int count = 0;

foreach (var i in select)

{

count++;

Console.WriteLine(count);

}

((ICookable)own2).Check();

((ICookable)own2).Cook();

foreach (var a in owns)

{

try

{

if (a.status == Own.Status.finish || a.status == Own.Status.ready)

{

throw new OwnNotWorking("Exception");

}

}

catch (OwnNotWorking ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

}

# 32. Паттерны проектирования. Паттерн Singleton.

Позволяет создать только один объект

public class App

{

public color col { get; set; }

public SizeOfText size { get; set; }

public void setColor(string colorName)

{

col = color.getInstance(colorName);

}

public void setSize(string sizeText)

{

size = SizeOfText.getInstance(sizeText);

}

}

public class color

{

private static color instance;

public string \_color { get; private set; }

protected color(string \_color)

{

this.\_color = \_color;

}

 public static color getInstance(string \_color)

{

if (instance == null)

instance = new color(\_color);

return instance;

}

}

public class SizeOfText

{

private static SizeOfText instance;

public string size { get; private set; }

protected SizeOfText(string name)

{

this.size = name;

}

 public static SizeOfText getInstance(string name)

{

if (instance == null)

instance = new SizeOfText(name);

return instance;

}

}

}

Main:

Console.WriteLine("\n\nЗадание 2");

App app = new();

app.setSize("12");

app.setColor("red");

Console.WriteLine("Цвет шрифта: " + app.col.\_color);

Console.WriteLine("Размер шрифта: " + app.size.size);

//попытка изменить

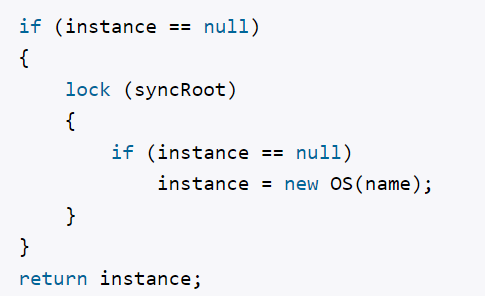
app.col = color.getInstance("white");

app.size = SizeOfText.getInstance("26");

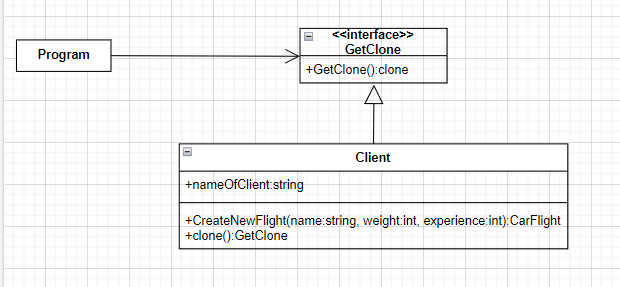
Console.WriteLine("Цвет шрифта: " + app.col.\_color);

Console.WriteLine("Размер шрифта: " + app.size.size);

В потокобезопасной:



# 33. Паттерны проектирования. Паттерн Prototype.



Просто перегружает метод clone() для каждого класса

# 34. Паттерны проектирования. Паттерн Builder.

Создать сложный объект из нескольких простых

//мука

class Flour

{

// какого сорта мука

public string Sort { get; set; }

}

// соль

class Salt

{ }

// пищевые добавки

class Additives

{

public string Name { get; set; }

}

class Bread

{

// мука

public Flour Flour { get; set; }

// соль

public Salt Salt { get; set; }

// пищевые добавки

public Additives Additives { get; set; }

public override string ToString()

{

StringBuilder sb = new StringBuilder();

if (Flour != null)

sb.Append(Flour.Sort + "\n");

if (Salt != null)

sb.Append("Соль \n");

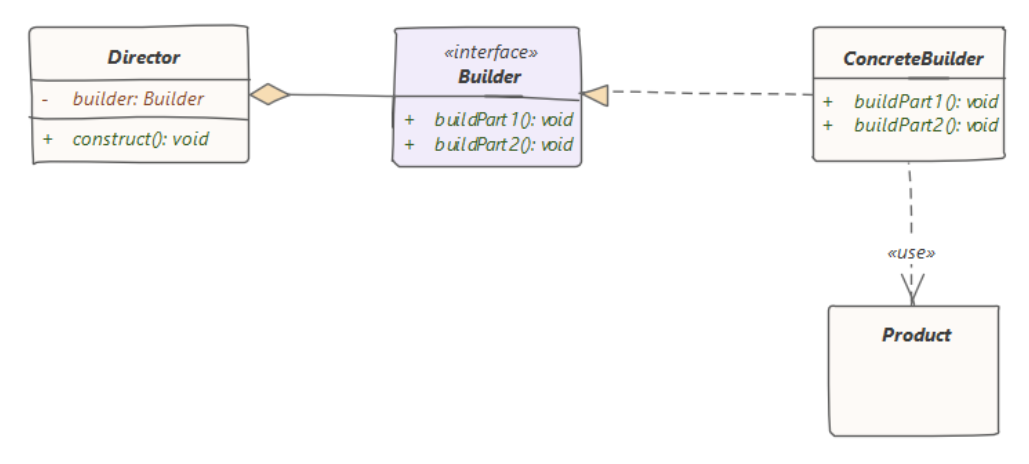
if (Additives != null)

sb.Append("Добавки: " + Additives.Name + " \n");

return sb.ToString();

}

}



# 35. Паттерны проектирования. Паттерн Adapter.

Изменяет функциональность интерфейса для конкретного класса

class Program

{

static void Main(string[] args)

{

// путешественник

Driver driver = new Driver();

// машина

Auto auto = new Auto();

// отправляемся в путешествие

driver.Travel(auto);

// встретились пески, надо использовать верблюда

Camel camel = new Camel();

// используем адаптер

ITransport camelTransport = new CamelToTransportAdapter(camel);

// продолжаем путь по пескам пустыни

driver.Travel(camelTransport);

Console.Read();

}

}

interface ITransport

{

void Drive();

}

// класс машины

class Auto : ITransport

{

public void Drive()

{

Console.WriteLine("Машина едет по дороге");

}

}

class Driver

{

public void Travel(ITransport transport)

{

transport.Drive();

}

}

// интерфейс животного

interface IAnimal

{

void Move();

}

// класс верблюда

class Camel : IAnimal

{

public void Move()

{

Console.WriteLine("Верблюд идет по пескам пустыни");

}

}

// Адаптер от Camel к ITransport

class CamelToTransportAdapter : ITransport

{

Camel camel;

public CamelToTransportAdapter(Camel c)

{

camel = c;

}

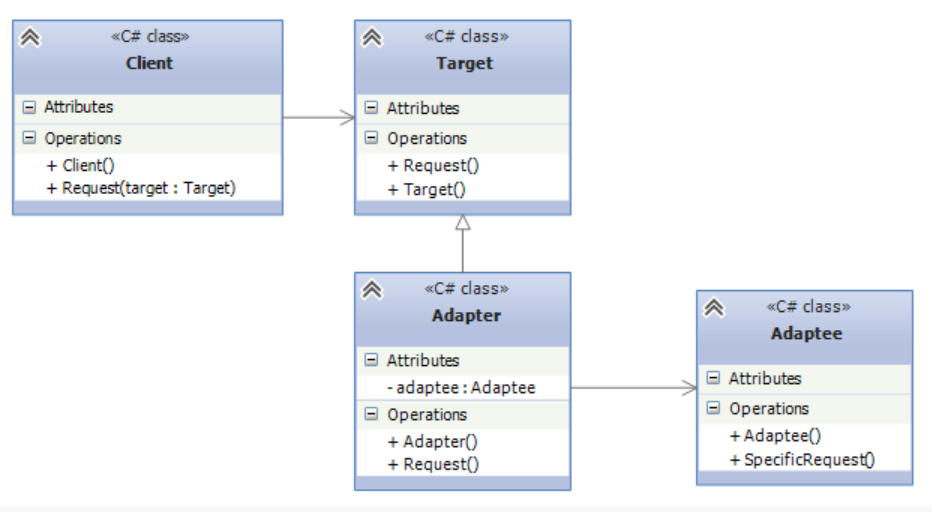
public void Drive()

{

camel.Move();

}

}



class Client

{

public void Request(Target target)

{

target.Request();

}

}

// класс, к которому надо адаптировать другой класс

class Target

{

public virtual void Request()

{}

}

// Адаптер

class Adapter : Target

{

private Adaptee adaptee = new Adaptee();

public override void Request()

{

adaptee.SpecificRequest();

}

}

// Адаптируемый класс

class Adaptee

{

public void SpecificRequest()

{}

}

# 36. Паттерны проектирования. Паттерн Decorator.

Позволяет динамически подключать к объекту дополнительную функциональность

class Program

{

static void Main(string[] args)

{

Pizza pizza1 = new ItalianPizza();

pizza1 = new TomatoPizza(pizza1); // итальянская пицца с томатами

Console.WriteLine("Название: {0}", pizza1.Name);

Console.WriteLine("Цена: {0}", pizza1.GetCost());

Pizza pizza2 = new ItalianPizza();

pizza2 = new CheesePizza(pizza2);// итальянская пиццы с сыром

Console.WriteLine("Название: {0}", pizza2.Name);

Console.WriteLine("Цена: {0}", pizza2.GetCost());

Pizza pizza3 = new BulgerianPizza();

pizza3 = new TomatoPizza(pizza3);

pizza3 = new CheesePizza(pizza3);// болгарская пиццы с томатами и сыром

Console.WriteLine("Название: {0}", pizza3.Name);

Console.WriteLine("Цена: {0}", pizza3.GetCost());

Console.ReadLine();

}

}

abstract class Pizza

{

public Pizza(string n)

{

this.Name = n;

}

public string Name { get; protected set; }

public abstract int GetCost();

}

class ItalianPizza : Pizza

{

public ItalianPizza() : base("Итальянская пицца")

{ }

public override int GetCost()

{

return 10;

}

}

class BulgerianPizza : Pizza

{

public BulgerianPizza()

: base("Болгарская пицца")

{ }

public override int GetCost()

{

return 8;

}

}

abstract class PizzaDecorator : Pizza

{

protected Pizza pizza;

public PizzaDecorator(string n, Pizza pizza) : base(n)

{

this.pizza = pizza;

}

}

class TomatoPizza : PizzaDecorator

{

public TomatoPizza(Pizza p)

: base(p.Name + ", с томатами", p)

{ }

public override int GetCost()

{

return pizza.GetCost() + 3;

}

}

class CheesePizza : PizzaDecorator

{

public CheesePizza(Pizza p)

: base(p.Name + ", с сыром", p)

{ }

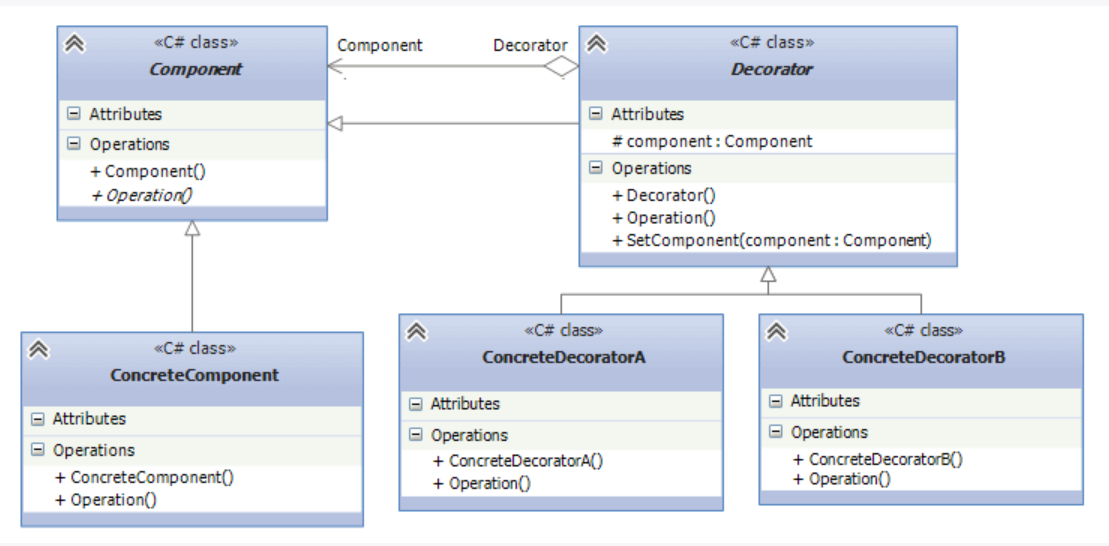
public override int GetCost()

{

return pizza.GetCost() + 5;

}

}



# 37. Паттерны проектирования. Паттерн State.

Изменяет состояние объекта

class Program

{

static void Main(string[] args)

{

Water water = new Water(WaterState.LIQUID);

water.Heat();

water.Frost();

water.Frost();

Console.Read();

}

}

enum WaterState

{

SOLID,

LIQUID,

GAS

}

class Water

{

public WaterState State { get; set; }

public Water(WaterState ws)

{

State = ws;

}

public void Heat()

{

if (State == WaterState.SOLID)

{

Console.WriteLine("Превращаем лед в жидкость");

State = WaterState.LIQUID;

}

else if (State == WaterState.LIQUID)

{

Console.WriteLine("Превращаем жидкость в пар");

State = WaterState.GAS;

}

else if (State == WaterState.GAS)

{

Console.WriteLine("Повышаем температуру водяного пара");

}

}

public void Frost()

{

if (State == WaterState.LIQUID)

{

Console.WriteLine("Превращаем жидкость в лед");

State = WaterState.SOLID;

}

else if (State == WaterState.GAS)

{

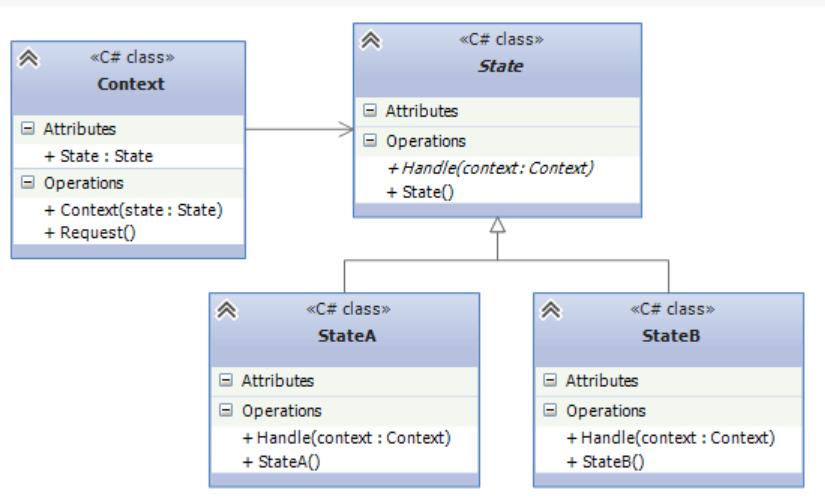
Console.WriteLine("Превращаем водяной пар в жидкость");

State = WaterState.LIQUID;

}

}

}



# 38. Паттерны проектирования. Паттерн Memento.

Сохраняет состояние объекта для дальнейшего восстановления

class Program

{

static void Main(string[] args)

{

Hero hero = new Hero();

hero.Shoot(); // делаем выстрел, осталось 9 патронов

GameHistory game = new GameHistory();

game.History.Push(hero.SaveState()); // сохраняем игру

hero.Shoot(); //делаем выстрел, осталось 8 патронов

hero.RestoreState(game.History.Pop());

hero.Shoot(); //делаем выстрел, осталось 8 патронов

Console.Read();

}

}

// Originator

class Hero

{

private int patrons = 10; // кол-во патронов

private int lives = 5; // кол-во жизней

public void Shoot()

{

if (patrons > 0)

{

patrons--;

Console.WriteLine("Производим выстрел. Осталось {0} патронов", patrons);

}

else

Console.WriteLine("Патронов больше нет");

}

// сохранение состояния

public HeroMemento SaveState()

{

Console.WriteLine("Сохранение игры. Параметры: {0} патронов, {1} жизней", patrons, lives);

return new HeroMemento(patrons, lives);

}

// восстановление состояния

public void RestoreState(HeroMemento memento)

{

this.patrons = memento.Patrons;

this.lives = memento.Lives;

Console.WriteLine("Восстановление игры. Параметры: {0} патронов, {1} жизней", patrons, lives);

}

}

// Memento

class HeroMemento

{

public int Patrons { get; private set; }

public int Lives { get; private set; }

public HeroMemento(int patrons, int lives)

{

this.Patrons = patrons;

this.Lives = lives;

}

}

// Caretaker

class GameHistory

{

public Stack<HeroMemento> History { get; private set; }

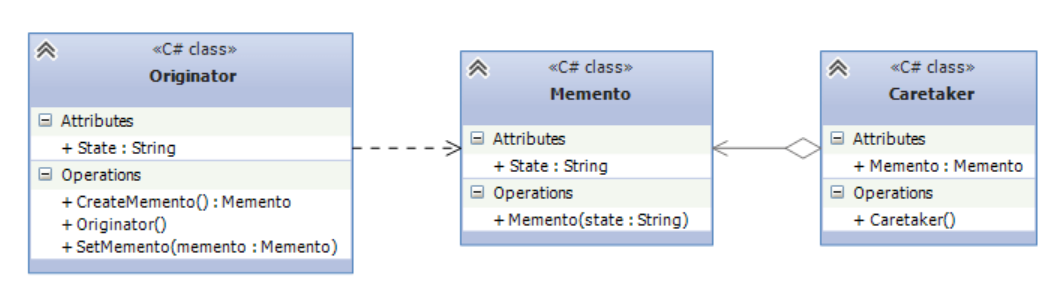
public GameHistory()

{

History = new Stack<HeroMemento>();

}

}



# 38. Паттерны проектирования. Паттерн Proxy.

Создает объект-заместитель, который может выступать в роли другого объекта и замещать его

class Client

{

void Main()

{

Subject subject = new Proxy();

subject.Request();

}

}

abstract class Subject

{

public abstract void Request();

}

class RealSubject : Subject

{

public override void Request()

{ }

}

class Proxy : Subject

{

RealSubject realSubject;

public override void Request()

{

if (realSubject == null)

realSubject = new RealSubject();

realSubject.Request();

}

}

