namespace \_3\_2

{

public class Location

{

public int lat { get; set; }

public int longg { get; set; }

public int speed { get; set; }

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

{

this.lat = lat;

this.longg = longg;

this.speed = speed;

}

}

public class Taxi

{

public string number { get; set; }

public Location location { get; set; }

public enum Status

{

busy,

free

}

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 $"{number} {location} {status}";

}

}

public class Park<T> where T: Taxi

{

List<Taxi> park = new List<Taxi>();

public void Add(Taxi obj)

{

park.Add(obj);

}

public void Del(Taxi obj)

{

park.Remove(obj);

}

public void Clear ()

{

park.Clear();

}

public string Find(Predicate<T> predicate)

{

foreach(T i in park)

{

if (predicate(i))

{

return i.number;

}

}

return null;

}

public override string ToString()

{

return $"{park}";

}

}

class Program

{

static void Main(string[] args)

{

Location location1 = new Location(14, 67, 60);

Location location2 = new Location(34, 56, 65);

Location location3 = new Location(58, 89, 70);

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

Taxi taxi1 = new Taxi("546", location1, Taxi.Status.busy);

Taxi taxi2 = new Taxi("457", location2, Taxi.Status.free);

Taxi taxi3 = new Taxi("523", location1, Taxi.Status.free);

Taxi taxi4 = new Taxi("137", location3, Taxi.Status.busy);

uber.Add(taxi1);

uber.Add(taxi2);

uber.Add(taxi3);

uber.Add(taxi4);

Console.WriteLine(uber.ToString());

Predicate<Taxi> predicate = (Taxi tax) => { return tax.status == Taxi.Status.free; };

Console.WriteLine(uber.Find(predicate));

}

}

}

namespace \_3\_3

{

public class SomeString : IComparer<SomeString>

{

public string s;

public SomeString(string s)

{

this.s = s;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

SomeString str = (SomeString)obj;

return (this.s.Length == str.s.Length && this.s[0] == str.s[0] && this.s.Substring(this.s.Length - 1) == str.s.Substring(str.s.Length - 1));

}

public int Compare(SomeString s1, SomeString s2)

{

if (s1.s.Length > s2.s.Length)

return 1;

else if (s1.s.Length < s2.s.Length)

return -1;

else return 0;

}

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

{

return new SomeString(s1.s + a1);

}

public static SomeString operator - (SomeString s2, char a2)

{

try

{

if (s2.s == null)

throw new Exception("Str is empty");

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

return new SomeString(s2.s = s2.s.Remove(0, 1));

}

}

public static class StringExtention

{

public static int Count(this SomeString str)

{

int count = 0;

foreach (var a in str.s)

{

if (a == ' ')

{

count++;

}

}

return count;

}

public static string Remove(this SomeString str)

{

foreach (var a in str.s)

{

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

{

str.s = str.s.Replace(a, ' ');

}

}

return str.s;

}

}

class Program

{

static void Main(string[] args)

{

string way = @"D:\СЕССИЯ\ООП\3-3\text.txt";

using (StreamWriter stream = new StreamWriter(way, 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 + 'a';

s2 = s2 - ' ';

stream.WriteLine(s1.s);

stream.WriteLine(s2.s);

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

stream.WriteLine(StringExtention.Count(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 s in select)

{

sum += s.Count();

}

stream.WriteLine(sum);

}

}

}

}

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 double w;

public double h;

public Button(string caption, int x, int y, double w, double 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} Startpoint: x = {startpoint.x} y = {startpoint.y} Width: {w} Height: {h}";

}

public override bool Equals(object obj)

{

if (obj == null) return false;

if (obj.GetType() != this.GetType()) return false;

Button button = (Button)obj;

return this.caption == button.caption && this.w == button.w && this.h == button.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(double q)

{

this.w = this.w \* q;

this.h = this.h \* q;

}

double square;

public double Square()

{

square = w \* h;

return square;

}

}

public class CheckButton

{

public State state;

public enum State

{

check = 1,

uncheck

}

}

public class User

{

public int Click { get; set; }

public int Resize { get; set; }

}

class Program

{

static void Main(string[] args)

{

Button button1 = new Button("try", 12, 45, 12.2, 12.7, CheckButton.State.check);

Button button2 = new Button("catch", 34, 5, 11.6, 7.8, CheckButton.State.uncheck);

Button button3 = new Button("finally", 6, 13, 5.6, 7.9, CheckButton.State.uncheck);

User user = new User();

Console.WriteLine(button1.ToString());

Console.WriteLine(button2.ToString());

Console.WriteLine(button3.ToString());

button1.Check();

button2.Check();

button3.Zoom(0.4);

Console.WriteLine(button1.Equals(button2));

Console.WriteLine(button1.ToString());

Console.WriteLine(button2.ToString());

Console.WriteLine(button3.ToString());

LinkedList<Button> list = new LinkedList<Button>();

list.AddFirst(button1);

list.AddFirst(button2);

list.AddFirst(button3);

foreach (var i in list)

{

Console.WriteLine(i);

}

button1.w = button1.Square();

button2.w = button2.Square();

button3.w = button3.Square();

Console.WriteLine(button1.w);

Console.WriteLine(button2.w);

Console.WriteLine(button3.w);

double z = Convert.ToDouble(Console.ReadLine());

var select = from i in list

where i.w == z

select i;

foreach (var i in select)

{

Console.WriteLine(i.w);

}

}

}

}

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

{

class Program

{

interface IScore

{

int Amount { get; set; }

int AddSum(int money);

int RemSum(int money);

}

abstract class Human

{

public string DateofBirth { get; set; }

}

class Person : Human, IScore

{

public int Amount { get; set; } = 0;

public int AddSum(int money)

{

return Amount += money;

}

public int RemSum(int money)

{

return Amount -= money;

}

static int count;

public Person(string date){

DateofBirth = date;

count++;

}

static Person()

{

Console.WriteLine("Статический конструктор");

}

public static int CountObj()

{

return count;

}

public override bool Equals(object obj)

{

if (obj == null) return false;

Person pers = (Person)obj;

return (this.DateofBirth == pers.DateofBirth);

}

}

class Bank : List<Person>

{

}

static void Main(string[] args)

{

Person person1 = new Person("17.02.2000");

Person person2 = new Person("17.03.2000");

Person person3 = new Person("17.04.2000");

Person person4 = new Person("17.02.2000");

Console.WriteLine(Person.CountObj());

person1.AddSum(10);

person1.RemSum(4);

Console.WriteLine(person1.Amount);

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

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

Bank belarus = new Bank();

belarus.Add(person1);

belarus.Add(person2);

belarus.Add(person3);

Bank alfa = new Bank();

alfa.Add(person2);

alfa.Add(person3);

alfa.Add(person4);

Bank vtb = new Bank();

vtb.Add(person1);

vtb.Add(person3);

vtb.Add(person4);

}

}

}

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

{

public interface IAction<T>

{

void Add(T obj);

void Remove(T obj);

void Clear();

void Info();

}

class NullSizeCollection : SystemException

{

public NullSizeCollection(string message):base(message)

{

Console.WriteLine("Коллекция пуста");

}

}

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

{

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

public List<T> Ts

{

get

{

return ts;

}

}

void IAction<T>.Add(T obj)

{

ts.Add(obj);

}

void IAction<T>.Remove(T obj)

{

try

{

if (ts.Count() == 0)

{

throw new NullSizeCollection("Коллекция пуста");

}

else

ts.Remove(obj);

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

void IAction<T>.Clear()

{

try

{

if (ts.Count() == 0)

{

throw new NullSizeCollection("Коллекция пуста");

}

else

ts.Clear();

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

void IAction<T>.Info()

{

foreach (var i in ts)

{

Console.WriteLine(i.ToString());

}

}

}

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("Anna", "Math", 8);

Student student2 = new Student("Katya", "History", 7);

Student student3 = new Student("Vlad", "OAP", 3);

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

IAction<Student> action = examcard;

action.Add(student1);

action.Add(student2);

action.Add(student3);

action.Info();

//action.Remove(student1);

//action.Info();

var selectbymark = from i in examcard.Ts

where i.Mark >= 4

select i;

foreach (var i in selectbymark)

{

Console.WriteLine(i.ToString());

}

double agv = 0;

int count = 0;

foreach (var i in selectbymark )

{

agv += i.Mark;

count++;

}

agv = agv / count;

Console.WriteLine(agv);

}

}

}

namespace karta

{

class PinErrorException : Exception

{

public PinErrorException(string message) : base(message)

{

Console.WriteLine("Пароль введен неверно");

}

}

public class CreditCard: ICreditCard

{

public int balance;

public int number;

readonly int pin1;

readonly int pin2;

public int parm = 0;

public CreditCard(int balance, int number, int pin1, int pin2)

{

this.balance = balance;

this.number = number;

this.pin1 = pin1;

this.pin2 = pin2;

}

public void CheckBalabce()

{

while (true)

{

if (parm <3)

{

try

{

Console.WriteLine("Введите pin1: ");

int pin1\_1 = Convert.ToInt32(Console.ReadLine());

if (pin1\_1 == pin1)

{

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

return;

}

else

{

parm++;

throw new PinErrorException("Неверный пароль");

}

}

catch (PinErrorException ex)

{

Console.WriteLine(ex.Message);

}

}

else

{

try

{

Console.WriteLine("Введите pin2: ");

int pin2\_2 = Convert.ToInt32(Console.ReadLine());

if (pin2\_2 == pin2)

{

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

return;

}

else

{

throw new PinErrorException("Неверный пароль");

}

}

catch (PinErrorException ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

void ICreditCard.Add(int obj)

{

balance = this.balance + obj;

Console.WriteLine("Баланс пополнен на: " + obj);

}

void ICreditCard.Get(int obj)

{

try

{

if (balance - obj > 0)

{

balance = this.balance - obj;

Console.WriteLine("С карты сняли: " + obj);

}

else

{

throw new Exception("Недостаточно средств");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

interface ICreditCard

{

void Add(int obj);

void Get(int obj);

}

class Program

{

static void Main(string[] args)

{

CreditCard creditcard = new CreditCard(1200, 45600, 1234, 4321);

CreditCard creditcard1 = new CreditCard(1560, 425670, 2234, 4322);

CreditCard creditcard2 = new CreditCard(30, 412600, 3234, 4323);

creditcard.CheckBalabce();

((ICreditCard)creditcard).Add(45);

((ICreditCard)creditcard).Get(400);

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

creditcards.Add(creditcard);

creditcards.Add(creditcard1);

creditcards.Add(creditcard2);

var selectbymoney = from s in creditcards

where s.balance > 100 && s.number.ToString().Contains("2") && s.number.ToString().Contains("0")

select s.balance;

int sum = 0;

foreach (var s in selectbymoney)

sum = +s;

Console.WriteLine(sum);

}

}

namespace Kosh

{

[Serializable]

class AddException : Exception

{

public AddException(string message) : base(message)

{

Console.WriteLine(message);

}

}

class DelException : Exception

{

public DelException(string message) : base(message)

{

Console.WriteLine(message);

}

}

interface INumber

{

int Number { get; set; }

}

public class Bill : INumber

{

public Bill(int n)

{

Number = n;

}

public int number;

public int Number

{

get { return number; }

set

{

if (value != 5 && value != 10 && value != 20 && value != 30 && value != 40)

Console.WriteLine("Ошибка");

else

number = value;

}

}

}

public class Wallet<T> where T : Bill

{

public List<Bill> wallet = new List<Bill>();

public static int count = 0;

public int AddToList(Bill obj)

{

wallet.Add(obj);

count++;

if (count < 200)

{

return count;

}

else

{

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

}

}

public void DeleteFromList()

{

if (count > 0)

{

foreach (Bill a in wallet)

{

var delete = from Bill in wallet orderby Bill.number select Bill;

wallet.Remove(wallet.First());

}

}

else

{

throw new DelException("Нет купюр в кошельке");

}

}

public void Count()

{

var count = from i in wallet group i by i.Number;

foreach (var i in count)

{

Console.WriteLine($"{i.Key} {i.Count()}");

}

}

}

class Program

{

static void Main(string[] args)

{

try

{

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

wallet.AddToList(new Bill(40));

wallet.AddToList(new Bill(10));

wallet.AddToList(new Bill(40));

wallet.AddToList(new Bill(20));

wallet.AddToList(new Bill(20));

wallet.AddToList(new Bill(40));

Console.WriteLine(wallet.ToString());

wallet.Count();

}

catch(AddException err)

{

Console.WriteLine(err.Message);

}

catch (DelException err)

{

Console.WriteLine(err.Message);

}

}

}

}

namespace pucik

{

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 base.ToString() + " " + name + " " + ID + " " + price;

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public void OnSale()

{

this.price -= this.price \* 0.7;

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);

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

queue.Enqueue(item1);

queue.Enqueue(item2);

queue.Enqueue(item3);

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();

}

}

}

namespace roma

1-2. Сделать абстрактный класс transport с вашей реализацией (я просто сделал string name). Наследовать его в классе Air.Добавить свойства CountOfPassengers и Speed,

а так же Status, который принимает значение из перечисления в классе с состояниями fly, ready, stop. Сделать интерфейс IAirable с методами Check() и Fly() и Наследовать его в Air.

Метод Check():

Если CountOfPassengers = 0 и Speed = 0, то Status = stop;

Если CountOfPassengers > 0 и Speed = 0, то Status = ready;

Если CountOfPassengers > 0, Speed > 0 и Status = ready, то Status = fly. Метод Fly() выводит Flying, если Status = fly, если нет - выбрасывает исключение

(можешь хоть базовое, я с сообщением делал). Продемонстрировать работу с объектом Air.

3. Сделать ВЕСЬ вывод еще и в файл.

4. Сделать интерфейс IAir... с таким же методом Check() и наследовать в Air. Метод из IAir... Должен выводить "Ready", если CountOfPassengers > 20 и <100. Продемонстрировать

оба метода в программе. (2 интерфейса, 2 метода но с 1 названием (IAir...Check(), IAirable.Check())).

5. 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.

{

public interface IAirble

{

void Check();

void Fly();

}

public interface IAirable2

{

void Check();

}

internal enum Status { fly = 1, ready, stop}

public abstract class Transport

{

public string Name;

}

class Air : Transport, IAirble, IAirable2

{

public int CountOfPassengers { get; set; }

public int Speed { get; set; }

private Status status;

public Status Status { get => status; }

public Air(string \_name, int \_CountOfPassengers, int \_speed, Status \_status)

{

this.Name = \_name;

this.CountOfPassengers = \_CountOfPassengers;

this.Speed = \_speed;

this.status = \_status;

}

public void Check()

{

if (CountOfPassengers == 0 && Speed == 0)

this.status = Status.stop;

if (CountOfPassengers > 0 && Speed == 0)

this.status = Status.ready;

if (CountOfPassengers > 0 && Speed > 0)

this.status = Status.fly;

}

public void Fly()

{

try

{

if (this.status == Status.fly)

Console.WriteLine("Flying");

else

Console.WriteLine("Not flying");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

void IAirable2.Check()

{

if (CountOfPassengers > 20 && CountOfPassengers < 100)

Console.WriteLine("Ready");

}

public override string ToString()

{

return base.ToString() + " " + Name + " " + CountOfPassengers + " " + Speed + " " + status;

}

}

class Program

{

static void Main(string[] args)

{

string way = @"D:\СЕССИЯ\ООП\8-3\text.txt";

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

{

Air air1 = new Air("A167", 37, 0, Status.ready);

Air air2 = new Air("B568", 0, 0, Status.stop);

Air air3 = new Air("C569", 245, 164, Status.fly);

Air air4 = new Air("D366", 234, 0, Status.ready);

Air air5 = new Air("E355", 59, 160, Status.fly);

sw.WriteLine(air1.ToString());

sw.WriteLine(air2.ToString());

sw.WriteLine(air3.ToString());

air1.Check();

air2.Check();

air1.Fly();

air3.Fly();

IAirable2 airable = air1;

IAirable2 airable2 = air3;

airable.Check();

airable2.Check();

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

list.Add(air1);

list.Add(air2);

list.Add(air3);

list.Add(air4);

list.Add(air5);

var select = from i in list

where i.Status == Status.fly

select i;

int agv = 0;

int count = 0;

foreach (var i in select)

{

agv += i.Speed;

count++;

}

agv = agv / count;

sw.WriteLine($"Cредняя скорость: {agv}");

}

}

}

//1

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 ConsoleApp1

{

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, "Чёрный");

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 void Print()

{

Console.WriteLine($"Координаты{X},{Y} Длинна{L},Ширина {H}, Цвет {Color}");

}

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

{

A.H = A.H + i;

A.L = A.L + i;

return A;

}

public override string ToString()

{

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

}

public int Sqr(Rectangle pl)

{

return pl.H \* pl.L;

}

}

}

//

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Runtime.Serialization.Formatters.Binary;

namespace Practice\_Exam\_2\_Repited

{

[Serializable]

public class User : IComparable

{

private string email;

private int password;

public string status;

public string Status

{

set

{

if(value == "signin" || value == "signout")

{

status = value;

}

else

{

throw new Exception();

}

}

get

{

return status;

}

}

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

{

this.email = email;

this.password = password;

this.status = status;

}

public int CompareTo(object o)

{

User a = o as User;

if(a.password > this.password)

{

return 1;

}

else if(a.password < this.password)

{

return -1;

}

else

{

return 0;

}

}

public override string ToString()

{

return ("МЭЙЛ " + email + " ПАРОЛЬ " + password + " СТАТУС " + status);

}

public override int GetHashCode()

{

return (password - 235);

}

public override bool Equals(object obj)

{

User user1 = obj as User;

if (user1 != null)

{

if (user1.email == this.email)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

}

[Serializable]

public class WedLet<T>

{

public static LinkedList<T> listic = new LinkedList<T>();

public void Add(T obj)

{

listic.AddLast(obj);

}

public void Remove(T obj)

{

listic.Remove(obj);

}

public int Count()

{

return listic.Count();

}

}

class Program

{

static void Main(string[] args)

{

try

{

User user1 = new User("qwert", 1111, "signout");

User user2 = new User("asdfg", 2222, "signin");

User user3 = new User("zxcbv", 3333, "signout");

User user4 = new User("asdghkj234", 4444, "signin");

User user5 = new User("asd123", 5555, "signin");

Console.WriteLine(user2.ToString());

Console.WriteLine(user1.ToString());

Console.WriteLine(user3.ToString());

Console.WriteLine(user4.ToString());

Console.WriteLine(user5.ToString());

Console.WriteLine(user5.CompareTo(user1));

WedLet<User> github = new WedLet<User>();

github.Add(user1);

github.Add(user2);

github.Add(user3);

github.Add(user4);

github.Add(user5);

var linq1 = from s in WedLet<User>.listic

where s.status == "signin"

select s;

Console.WriteLine(linq1.Count());

var serialize = new BinaryFormatter();

using (var file = new FileStream("file.bin", FileMode.OpenOrCreate))

{

serialize.Serialize(file, github);

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

}

}

//Iuseble

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Threading.Channels;

namespace UserList\_Dictionary

{

public interface IUseable

{

void Add(int key, User value);

bool Delete(int key);

void Find(int key);

}

[Serializable]

public class User

{

public string Name { get; set; }

public string Number { get; set; }

public int Tarif { get; set; }

public User()

{

}

public User(string name, string number, int tarif)

{

Name = name;

Number = number;

Tarif = tarif;

}

}

public class UserList : IUseable

{

public Dictionary<int, User> list;

public UserList()

{

list = new Dictionary<int, User>();

}

public void Print()

{

foreach (var lis in list)

Console.WriteLine($" Ключ {lis.Key} Имя абонента: {lis.Value.Name} Номер {lis.Value.Number}\n");

}

public void Add(int key, User value)

{

list.Add(key, value);

if (key == 0)

throw new NullReferenceException();

}

public bool Delete (int key)

{

return list.Remove(key);

}

public void Find(int key)

{

bool i = false;

foreach (var lis in list)

{

if (lis.Key == key)

{

Console.WriteLine($"{lis.Value.Number }");

i = true;

}

}

if (i == false)

throw new UserNotFound("Пользователь не найден ");

}

}

class UserNotFound : Exception

{

public UserNotFound(string message)

: base(message) { }

}

class Program

{

static void Main(string[] args)

{

try

{

UserList list = new UserList();

list.Add(1, new User("Lova", "375447270499", 1));

list.Add(2, new User("Jony", "375498772356", 3));

list.Add(3, new User("Lera", "375290120940", 2));

list.Add(4, new User("Misha", "375584770491", 1));

list.Delete(4);

list.Print();

list.Find(2);

Console.WriteLine(list);

//4

var number = list.list.Count(x => x.Value.Number.EndsWith('0')||x.Value.Number.EndsWith('1'));

Console.WriteLine(number);

//5

var newUser = new User();

var binaryFormatter = new BinaryFormatter();

using (var fileStream = new FileStream(@"binary.bin",FileMode.OpenOrCreate))

{

User userForSerializing = list.list[1];

binaryFormatter.Serialize(fileStream,userForSerializing);

fileStream.Position = 0;

newUser=(User) binaryFormatter.Deserialize(fileStream);

Console.WriteLine($"{newUser.Name} , {newUser.Number} , {newUser.Tarif}");

//Ох уж эти любители of Lova

}

}

catch (UserNotFound ex)

{

Console.WriteLine($"Ошибка: {ex.Message}");

}

}

}

}

//Card

using System.Runtime.InteropServices;

var Card = new Card();

Console.WriteLine(Card.Balance);

((ICard)Card).Add(1);

Console.WriteLine(Card.Balance);

((ICard)Card).Get(102);

Console.WriteLine(Card.Balance);

var card1 = new Card();

var card2 = new Card();

var card3 = new Card();

var card4= new Card();

var list = new List<Card>() { card1, card2, card3, card4 };

int sum = list.Where(x => x.Balance is >= 100 and <= 200 && x.number.Contains('9')).Sum(x=>x.Balance);

public interface ICard

{

void Add(int number);

int Get(int number);

}

public class Card:ICard

{

private int balance = 100;

public string number = "asaf9";

private readonly string pin2 = "1111";

private readonly string pin = "0000";

public int Balance

{

get

{

CheckPincode();

return balance;

}

set

{

CheckPincode();

balance=value;

}

}

void ICard.Add(int number)

{

Balance = balance + number;

}

int ICard.Get(int number)

{

try

{

if (number > balance)

throw new CanNotException("Balance less than number");

}

catch (CanNotException e)

{

Console.WriteLine(e);

return 0;

}

Balance = balance - number;

return number;

}

private void CheckPincode()

{

Console.WriteLine("Проверка пинкода");

bool isPincodeCorrect = false;

string currentPin = this.pin;

string userEntered;

for (int i = 0; i < 3 && isPincodeCorrect == false; i++)

{

try

{

userEntered = Console.ReadLine();

if (userEntered != currentPin)

{

throw new PinErrorException("Pin is wrong");

}

isPincodeCorrect = true;

}

catch(PinErrorException e)

{

using (var stream = new StreamWriter("Log.txt",true))

{

stream.WriteLine("Ошибка");

}

}

}

if (isPincodeCorrect)

{

Console.WriteLine("Проверка пинкода прошла");

return;

}

currentPin = this.pin2;

for (;;)

{

userEntered = Console.ReadLine();

if (userEntered == currentPin)

{

Console.WriteLine("Проверка пинкода прошла");

return;

}

}

}

}

internal class CanNotException : Exception

{

public CanNotException(string message)

: base(message) { }

}

public class PinErrorException :Exception

{

public PinErrorException(string message)

: base(message) { }

}