namespace Figure

{

interface IFigure

{

public void Print();

}

[Serializable]

public class Rectangle : IFigure

{

[NonSerialized]

private int x;

[NonSerialized]

private int y;

private int l;

private int h;

private string color="yellow"; // по умолчанию

public int X { get => x; set => x = value; }

public int Y { get => y; set => y = value; }

public int L { get => l; set => l = value; }

public int H { get => h; set => h = value; }

public string Color { get => color; set => color = value; }

public Rectangle()

{

}

public Rectangle(int x, int y, int l, int h)

{

this.X = x;

this.Y = y;

this.L = l;

this.H = h;

}

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

{

this.Color = color;

}

public void Print()

{

Console.WriteLine($"Координаты прямоугольника: {X}, {Y}; \n" +

$"Длина: {L}, ширина: {H}; \n" +

$"Цвет: {Color}.\n");

}

public override string ToString()

{

return ("Метод переопределён... ") + base.ToString();

}

public int Square()

{

return this.X \* this.Y;

}

public static Rectangle operator +(Rectangle obj1, Rectangle obj2)

{

return new Rectangle(obj1.X + obj2.X, obj1.Y + obj2.Y, obj1.L + obj2.L, obj1.H + obj2.H);

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rec = new Rectangle();

Rectangle rec1 = new Rectangle(20,10,15,20, "red");

Rectangle rec2 = new Rectangle(40, 50, 35, 80, "blue");

Rectangle rec3 = new Rectangle(10, 10, 10, 10);

Rectangle rec4 = new Rectangle(50, 60, 80, 2);

Rectangle rec5 = new Rectangle(90, 100, 75, 80, "black");

rec.Print();

rec1.Print();

rec2.Print();

rec3.Print();

rec4.Print();

var new\_rec = rec3 + rec4;

Console.WriteLine($"Новый прямоугольник:");

new\_rec.Print();

Console.WriteLine(rec1.ToString());

Console.WriteLine($"Площадь прямоугольника rec1 = {rec1.Square()}");

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

list.Add(rec);

list.Add(rec1);

list.Add(rec2);

list.Add(rec3);

list.Add(rec4);

list.Add(rec5);

Console.ForegroundColor = ConsoleColor.DarkMagenta;

Console.WriteLine("\n\nПроверка работы списка: ");

foreach (var n in list)

n.Print();

var w = list.OrderBy(t => t.X).ThenBy(r => r.Y).ThenBy(s => s.Square());

Console.WriteLine(w.First()); // первый отсортированный

Console.WriteLine(w.Last()); // последний отсортированный

Console.ForegroundColor = ConsoleColor.White;

//BinaryFormatter binaryFormatter = new BinaryFormatter();

//using (FileStream serial = new FileStream("rec.bin", FileMode.OpenOrCreate));

//{

// binaryFormatter.Serialize(serial, list);

//}

}

}

}

namespace roma

{

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:\Подливка\Сессия\2.0\ООП\Задачи\Innumerable\_9\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}");

}

}

}

}

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);

}

}

}

}

namespace OOP   
{   
  
class User : IComparable<User>   
{   
public readonly string login;   
public readonly string pasword;   
  
  
public User(string l, string p)   
{   
login = l;   
pasword = p;   
}   
//перегрузка открытых метов Object   
public override int GetHashCode()   
{   
return base.GetHashCode();   
}   
  
public override bool Equals(object obj)   
{   
return base.Equals(obj);   
}   
public override string ToString()   
{   
return base.ToString();   
}   
  
//перегрузка метода IComparable   
public int CompareTo(User obj)   
{   
int result = -1;   
if(obj!=null)   
{   
if (login == obj.login && pasword == obj.pasword)   
result = 1;   
}   
return result;   
}   
}   
class Program   
{   
static void Main(string[] args)   
{   
User user1 = new User("Dasha","556156847a");   
User user2 = new User("Vasya58", "8544");   
User user3 = new User("Vasya58", "8544");   
User user4 = new User("Dima1", "8544gs");   
User user5 = new User("Vika", "8er564644");   
  
LinkedList<User> linkedList = new LinkedList<User>();   
LinkedListNode<User> listNode = new LinkedListNode<User>(user1);   
linkedList.AddFirst(listNode);   
linkedList.AddAfter(listNode, user2);   
linkedList.AddAfter(listNode, user3);   
linkedList.AddAfter(listNode, user4);   
linkedList.AddAfter(listNode, user5);   
  
for(int i=0; i< linkedList.Count; i++)   
{   
Console.WriteLine(listNode.Value.login);   
listNode = [listNode.Next](https://vk.com/away.php?to=http%3A%2F%2FlistNode.Next&cc_key=);   
}   
  
IEnumerable<User> newCollection = linkedList.Where(n => (n.pasword.Length) < 8).Select(n => n).ToArray();   
  
Console.WriteLine("Отбор по паролям");   
foreach(User item in newCollection)   
{   
Console.WriteLine(item.login);   
}   
  
Console.WriteLine($"user1 vs user2 {user1.CompareTo(user2)}");   
Console.WriteLine($"user2 vs user3 {user2.CompareTo(user3)}");   
}   
}   
}

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 Podliva

{

class Item

{

public string Name { get; set; }

public int ID { get; set; }

public double Price { get; set; }

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

{

Name = name;

this.ID = ID;

Price = price;

}

public override string ToString()

{

return base.ToString() + " " + Name + " " + ID + " " + Price;

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public void Start()

{

this.Price = this.Price \* 0.5;

Console.WriteLine("sale");

}

}

class Manager

{

public delegate void Sale();

public event Sale sales;

public void SaleOn()

{

sales();

}

}

class Shop : IEnumerable

{

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

public Queue<Shop> shop = new Queue<Shop>();

public Queue<Item> GetQueue { get { return queue; } }

public void qAdd(Item obj)

{

queue.Enqueue(obj);

}

public void qRemove(Item obj)

{

queue.Dequeue();

}

public void qClear(Item obj)

{

queue.Clear();

}

public IEnumerator GetEnumerator()

{

return queue.GetEnumerator();

}

public static int operator +(Shop obj)

{

obj.sAdd(obj);

return 0;

}

public static int operator -(Shop obj)

{

obj.sRemove(obj);

return 0;

}

public void sAdd(Shop obj)

{

shop.Enqueue(obj);

}

public void sRemove(Shop obj)

{

shop.Dequeue();

}

}

class Program

{

static void Main(string[] args)

{

Item item1 = new Item("Диван", 1, 32);

Item item2 = new Item("Диван", 2, 40);

Item item3 = new Item("Диван", 3, 30);

Item item4 = new Item("Диван", 4, 36);

Shop que = new Shop();

que.qAdd(item1);

que.qAdd(item2);

que.qAdd(item3);

que.qAdd(item4);

Console.WriteLine(item1.ToString());

Console.WriteLine(item1.GetHashCode());

foreach (var item in que)

{

Console.WriteLine(item);

}

Manager manager = new Manager();

manager.sales += item1.Start;

manager.sales += item2.Start;

manager.sales += item3.Start;

manager.SaleOn();

foreach (var item in que)

{

Console.WriteLine(item);

}

string whatName = "Диван";

var itemCount = (from t in que.queue where t.Name == whatName select t).Count();

Console.WriteLine(itemCount);

}

}

}

namespace Podliva

{

class Sqr : Function

{

public int C { get; set; }

}

class Liner : Function

{

public double A { get; set; }

public double B { get; set; }

}

abstract class Function

{

public double 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 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;

}

}

}

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];

}

}

class Program

{

static void Main(string[] args)

{

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));

}

}

}

namespace Practise\_Exam\_Repited\_7

{

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();

}

}

}

}

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 Ex

{

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 = @"D:\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);

}

}

}

}

}

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 Exersise\_6

{

enum Form

{

our = 1,

your =2,

my = 3

}

class Company : IManage

{

string \_name;

int \_workerCount;

Form \_form;

int \_year1;

int \_year2;

int \_year3;

int \_year4;

public string Name

{

get => \_name;

set => \_name = value;

}

public int WorkerCount

{

get => \_workerCount;

set => \_workerCount = value;

}

public Form Formm

{

get => \_form;

set => \_form = value;

}

public int Year1

{

get => \_year1;

set => \_year1 = value;

}

public int Year2

{

get => \_year2;

set => \_year2 = value;

}

public int Year3

{

get => \_year3;

set => \_year3 = value;

}

public int Year4

{

get => \_year4;

set => \_year4 = value;

}

public (int max, int min) MaxMin()

{

(int Max, int Min) tuple = (0, \_year1);

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

vs.Add(\_year1);

vs.Add(\_year2);

vs.Add(\_year3);

vs.Add(\_year4);

foreach(int y in vs)

{

if(y > tuple.Max)

{

tuple.Max = y;

}

if(y < tuple.Min)

{

tuple.Min = y;

}

}

return tuple;

}

float IManage.MaxAvg()

{

float sum = 0;

float result;

sum = (float)(\_year1 + \_year2 + \_year3 + \_year4);

result = sum / 4;

return result;

}

public static Company operator ++(Company company)

{

company.\_workerCount++;

return company;

}

public static Company operator --(Company company)

{

if (company.\_workerCount > 0)

{

company.\_workerCount--;

return company;

}

throw new Exception("Ziro Exception");

}

public static Company operator +(Company company, int n)

{

company.\_workerCount = company.\_workerCount + n;

return company;

}

}

static class Extension

{

public static Company DeleteInfo(this Company company)

{

company.Year1 = 0;

company.Year2 = 0;

company.Year3 = 0;

company.Year4 = 0;

return company;

}

public static(double max1, double max2) GetMax(this List<Company> list)

{

(double max1, double max2) tuple;

tuple.max1 = ((IManage)list[0]).MaxAvg() / list[0].WorkerCount;

tuple.max2 = ((IManage)list[1]).MaxAvg() / list[1].WorkerCount;

double temp;

foreach (Company i in list)

{

temp = ((IManage)i).MaxAvg() / i.WorkerCount;

if (temp > tuple.max1)

{

tuple.max1 = temp;

}

}

foreach (Company i in list)

{

temp = ((IManage)i).MaxAvg() / i.WorkerCount;

if (temp > tuple.max2 && temp != tuple.max1)

{

tuple.max2 = temp;

}

}

return tuple;

}

}

interface IManage

{

float MaxAvg();

}

class Program

{

static void Main(string[] args)

{

Company company1 = new Company();

Form form1;

form1 = Form.my;

company1.Formm = form1;

company1.Name = "Pizdec";

company1.WorkerCount = 320;

company1.Year1 = 15724;

company1.Year2 = 16000;

company1.Year3 = 1000;

company1.Year4 = 724;

Company company2 = new Company();

Form form2;

form2 = Form.our;

company2.Formm = form2;

company2.Name = "Pizda";

company2.WorkerCount = 1000;

company2.Year1 = 25724;

company2.Year2 = 16000;

company2.Year3 = 1000;

company2.Year4 = 7240;

Console.WriteLine(company1.MaxMin());

Console.WriteLine(((IManage)company1).MaxAvg());

company1++;

Console.WriteLine(company1.WorkerCount);

company1--;

Console.WriteLine(company1.WorkerCount);

Console.WriteLine((company1 + 2).WorkerCount);

//company1.DeleteInfo();

Console.WriteLine($"{company1.Year1}, {company1.Year2}, {company1.Year3}, {company1.Year4}");

List<Company> companies = new List<Company>();

companies.Add(company1);

companies.Add(company2);

Console.WriteLine(companies.GetMax());

Console.ReadKey();

IEnumerable<Company> SortByFormul()

{

var request = from item in companies

where companies.GetMax() != (null,null) && item.Formm == Form.my

select item;

return request;

}

Console.WriteLine(SortByFormul());

}

}

}

namespace Practice\_Exam\_9

{

class Program

{

abstract class AbstractUser

{

public DateTime Date;

}

class User : AbstractUser

{

public string password;

public string login;

public string Password

{

set

{

if (value.Length < 6)

{

throw new Error1();

}

else if (value.Length > 12)

{

throw new Error2();

}

else

{

password = value;

}

}

get

{

return password;

}

}

public User(string login, string password, DateTime Date)

{

this.login = login;

Password = password;

this.Date = Date;

}

public override string ToString()

{

return (" LOGIN " + login + " PASSWORD " + password + " TIME " + Date);

}

}

//class listic<User>

//{

// List<User> list = new List<User>();

// public DateTime FindMin(User obj)

// {

// DateTime time1 = new DateTime();

// if (obj.Date < this.Date)

// {

// obj.Date = time1;

// return time1;

// }

// else

// {

// return obj.Date;

// }

// }

//}

class Error1 : Exception

{

public Error1() : base("ERROR ----- 2") { }

}

class Error2 : Exception

{

public Error2() : base("ERROR ----- 1") { }

}

static void Main(string[] args)

{

try

{

DateTime time1 = new DateTime(2001, 6, 27);

DateTime time2 = new DateTime(2011, 4, 16);

DateTime time3 = new DateTime(2020, 5, 15);

DateTime time4 = new DateTime(2009, 7, 8);

User user1 = new User("arsenii", "1242323", time1);

User user2 = new User("dima", "qweqweqw", time2);

User user3 = new User("dasha", "123asdsa2", time3);

User user4 = new User("valera", "1113123", time4);

Console.WriteLine(user1.ToString());

Console.WriteLine(user2.ToString());

Console.WriteLine(user3.ToString());

Console.WriteLine(user4.ToString());

List<User> listik = new List<User>();

listik.Add(user1);

listik.Add(user2);

listik.Add(user3);

listik.Add(user4);

//var linq1 = from user in listik

// where user.FindMin(user)

}

catch (Error1 e)

{

Console.WriteLine(e.Message);

}

catch (Error2 e)

{

Console.WriteLine(e.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 Practice\_Exam\_10

{

public interface IClearnable

{

void Clearn();

}

public enum specialization { poit, isit, web, mobile };

public class BSTUStudent

{

public string name;

public int group;

public specialization specialization;

public int mark1;

public int mark2;

public int mark3;

public int mark4;

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

{

this.name = name;

this.group = group;

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 + " Специальность " + specialization);

}

}

public class STGroup : 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("Arsenii", 6, specialization.poit, 6, 7, 8, 5);

BSTUStudent student2 = new BSTUStudent("Dima", 4, specialization.poit, 4, 5, 8, 8);

BSTUStudent student3 = new BSTUStudent("Dasha", 8, specialization.mobile, 4, 4, 4, 9);

BSTUStudent student4 = new BSTUStudent("Shyra", 2, specialization.web, 7, 7, 7, 7);

Console.WriteLine(student1.ToString());

var tuple = BSTUStudent.Getmarks(student1);

STGroup listic = new STGroup();

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();

foreach (BSTUStudent stud in listic.GetList)

{

Console.WriteLine(stud.name);

Console.WriteLine(stud.group);

Console.WriteLine(stud.specialization);

}

}

}

}

namespace ConsoleApp2

{

class Students

{

public int mark;

public string name;

public string subject;

public Students(string Name, int Mark, string Subject)

{

this.name = Name;

this.mark = Mark;

this.subject = Subject;

}

public Students()

{

}

public override string ToString()

{

return mark + " " + name + " " + subject;

}

}

interface IAction<T>

{

void Add(T a);

void Del(T a);

void Clean();

void Show();

}

class ExamCard<T> : IAction<T> where T : new()

{

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

public void Add(T a)

{

list.Add(a);

}

public void Del(T a)

{

try

{

if (list.Count == 0)

{

throw new NullSizeCollection("Коллекция пустая");

}

else list.Remove(a);

}

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 Show()

{

Console.WriteLine("Вся коллекция: ");

foreach (var l in list)

Console.WriteLine(l);

}

}

class NullSizeCollection : Exception

{

private string message;

public override string Message

{

get

{

return message;

}

}

public NullSizeCollection(string mess)

{

message = mess;

}

}

static class Met

{

public static void qwe(this Students st)

{

Random random = new Random();

st.mark += random.Next(1, 3);

}

}

class Program

{

static void Main(string[] args)

{

Students st1 = new Students("qwe", 6, "rer");

Students st2 = new Students("qwe", 7, "rer");

Students st3 = new Students("qwe", 5, "rer");

Students st4 = new Students("qwe", 3, "rer");

ExamCard<Students> st = new ExamCard<Students>();

((IAction<Students>)st).Add(st1);

((IAction<Students>)st).Add(st2);

((IAction<Students>)st).Add(st3);

((IAction<Students>)st).Add(st4);

((IAction<Students>)st).Show();

((IAction<Students>)st).Del(st3);

var linq1 = from s in ExamCard<Students>.list

where s.mark >= 4

select s;

Console.WriteLine(linq1.Count());

var linq2 = from s in ExamCard<Students>.list

select s.mark;

Console.WriteLine(linq2.Average());

st1.qwe();

Console.WriteLine(st1);

}

}

}

namespace Practice\_Exam\_9

{

class Program

{

abstract class AbstractUser

{

public DateTime Date;

}

class User : AbstractUser

{

public string password;

public string login;

public string Password

{

set

{

if(value.Length < 6)

{

throw new Error1();

}

else if (value.Length > 12)

{

throw new Error2();

}

else

{

password = value;

}

}

get

{

return password;

}

}

public User(string login, string password, DateTime Date)

{

this.login = login;

Password = password;

this.Date = Date;

}

public override string ToString()

{

return (" LOGIN " + login + " PASSWORD " + password + " TIME " + Date);

}

}

//class listic<User>

//{

// List<User> list = new List<User>();

// public DateTime FindMin(User obj)

// {

// DateTime time1 = new DateTime();

// if (obj.Date < this.Date)

// {

// obj.Date = time1;

// return time1;

// }

// else

// {

// return obj.Date;

// }

// }

//}

class Error1 : Exception

{

public Error1() : base("ERROR ----- 2") { }

}

class Error2 : Exception

{

public Error2() : base("ERROR ----- 1") { }

}

static void Main(string[] args)

{

try

{

DateTime time1 = new DateTime(2001, 6, 27);

DateTime time2 = new DateTime(2011, 4, 16);

DateTime time3 = new DateTime(2020, 5, 15);

DateTime time4 = new DateTime(2009, 7, 8);

User user1 = new User("arsenii", "1242323", time1);

User user2 = new User("dima", "qweqweqw", time2);

User user3 = new User("dasha", "123asdsa2", time3);

User user4 = new User("valera", "1113123", time4);

Console.WriteLine(user1.ToString());

Console.WriteLine(user2.ToString());

Console.WriteLine(user3.ToString());

Console.WriteLine(user4.ToString());

List<User> listik = new List<User>();

listik.Add(user1);

listik.Add(user2);

listik.Add(user3);

listik.Add(user4);

//var linq1 = from user in listik

// where user.FindMin(user)

}

catch(Error1 e)

{

Console.WriteLine(e.Message);

}

catch (Error2 e)

{

Console.WriteLine(e.Message);

}

}

}

}

namespace Innumerable\_10\_БГТУ\_Мое\_решение\_\_

{

enum Specialization

{

isit,

poibms,

poit,

deivi

}

class BSTUStudent

{

public override string ToString()

{

return ("Имя " + Name + " Группа " + Group + " Специальность " + Specialization);

}

public string Name {get; set;}

public int Group{get; set;}

public int Course {get; set;}

public Specialization Specialization { get; set;}

public int Mark1 { get; set; }

public int Mark2 { get; set; }

public int Mark3 { get; set; }

public int Mark4 { get; set; }

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

{

Specialization = specialization;

Name = name;

Group = group;

Course = course;

Mark1 = mark1;

Mark2 = mark2;

Mark3 = mark3;

Mark4 = mark4;

}

public static (int, int, double) Marks(BSTUStudent obj)

{

int[] marks = { obj.Mark1, obj.Mark2, obj.Mark3, obj.Mark4 };

int max = marks.Max();

int min = marks.Min();

double average = marks.Average();

var result = ( min, max, average);

return result;

}

}

public interface IClearnable

{

void lClearn();

}

class Groups : IClearnable

{

public List<BSTUStudent> Array = new List<BSTUStudent>();

public List<BSTUStudent> GetArray { get { return Array; } }

public void lAdd(BSTUStudent obj)

{

Array.Add(obj);

}

public void lClearn()

{

Array.Clear();

}

}

class Program

{

static void Main(string[] args)

{

BSTUStudent student1 = new BSTUStudent(Specialization.poibms, "Vlad", 8, 2, 4, 4, 6, 5 );

BSTUStudent student2 = new BSTUStudent(Specialization.poit, "Katya", 4, 3, 7, 4, 5, 5);

BSTUStudent student3 = new BSTUStudent(Specialization.poibms, "Nikita", 8, 2, 6, 5, 6, 5);

BSTUStudent student4 = new BSTUStudent(Specialization.poibms, "Kostya", 8, 2, 4, 4, 4, 4);

BSTUStudent.Marks(student1);

Groups mobilki = new Groups();

mobilki.lAdd(student1);

mobilki.lAdd(student2);

mobilki.lAdd(student3);

mobilki.lAdd(student4);

//var average = mobilki.Array.Select((Mark1, Mark2) => Mark1 + Mark2).Sum() / ratings.Sum();

//var maxAverage = (from t in mobilki.Array where t.Marks select t).Take(2);

//Console.WriteLine(maxAverage);

}

}

}

namespace Card\_pin\_pin2\_\_\_6\_1\_

{

class Program

{

public interface ICard

{

void Add(int num);

void Get(int num);

}

public class Card : ICard

{

private int balance;

private int number;

private readonly int pin;

private readonly int pin2;

public Card(int Pin, int Pin2, int Balance, int Number)

{

pin = Pin;

pin2 = Pin2;

balance = Balance;

number = Number;

}

private bool canBeAccesed = false;

public void ShowBalance()

{

for (int i=0; i<3; i++)

{

try

{

Console.WriteLine("Введите пароль: ");

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

if (trypin!=pin)

{

throw new PinError("Wrong password");

}

else

{

Console.WriteLine($"BALANCE: {balance}");

canBeAccesed = true;

return;

}

}

catch (Exception ex)

{

using (var file = new StreamWriter("bruh.txt", true))

{

file.WriteLine($"{ex.GetType().Name} {ex.Message} {DateTime.Now} {MethodInfo.GetCurrentMethod().Name} {this.ToString()}");

}

}

}

while (true)

{

Console.WriteLine("Введите повторный пароль: ");

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

if (trypin != pin2)

{

Console.WriteLine("Incorrect");

}

else

{

Console.WriteLine($"BALANCE: {balance}");

canBeAccesed = true;

return;

}

}

}

public void Add(int num)

{

if (canBeAccesed==true)

{

balance += num;

Console.WriteLine($"Баланс пополнен. Сумма на счету: {balance}");

}

else Console.WriteLine("No authoration");

}

public void Get(int num)

{

if (canBeAccesed == true)

{

if (num>balance)

throw new CanNotExeption("CAN'T");

else

{

balance -= num;

Console.WriteLine($"После списывания остаток на карте составляет: {balance}");

}

}

else Console.WriteLine("No authoration");

}

public int LinQ(List<Card> cards)

{

var selected = cards.Where(x => (x.balance > 100 && x.balance < 200) && (x.number.ToString().Contains("9"))).Select(x => x.balance);

return selected.Sum();

}

[Serializable]

internal class PinError : Exception

{

public PinError()

{

}

public PinError(string message) : base(message)

{

}

}

[Serializable]

internal class CanNotExeption : Exception

{

public CanNotExeption()

{

}

public CanNotExeption(string message) : base(message)

{

}

}

}

static void Main(string[] args)

{

var card = new Card(1111, 1111, 5000, 1);

card.ShowBalance();

card.Add(1200);

card.ShowBalance();

card.Get(5000);

List<Card> cards = new List<Card>();

cards.Add(new Card(1234, 4321, 500, 1));

cards.Add(new Card(1111, 1111, 1000, 2));

cards.Add(new Card(2222, 2222, 800, 3));

cards.Add(new Card(3333, 3333, 150, 9));

Console.WriteLine("\n\nСумма на счету у карты, у которой баланс [100;200] и присутствует цифра 9 в номере: ");

Console.Write(card.LinQ(cards));

Console.WriteLine();

}

}

}

namespace Bill\_4\_5\_

{

class Program

{

interface INumber

{

int Number { get; set; }

}

[Serializable]

public class Bill : INumber

{

private int number;

public int Number

{

get

{

return number;

}

set

{

if (value >= 0 && (value == 5 || value == 10 || value == 20 || value == 50 || value == 100))

number = value;

else

{

throw new Error();

}

}

}

public Bill(){}

public override string ToString()

{

return base.ToString();

}

public Bill(int number)

{

this.Number = number;

}

[Serializable]

class Error : Exception

{

public Error() : base("ERROR") { }

}

}

[Serializable]

public class Wallet<T> where T : Bill

{

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

public void Add(Bill obj)

{

if (money.Count > 200)

{

throw new ToMuchMoney();

}

else

{

money.Add((T)obj);

}

}

public void Remove()

{

if (money.Count < 0)

throw new NoMoney();

else

{

IEnumerable<T> delete = from i in money orderby i.Number select i;

money.Remove(delete.First());

}

}

public void Count()

{

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

foreach (var i in count)

{

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

}

}

internal class ToMuchMoney : Exception

{

public ToMuchMoney()

{

}

public ToMuchMoney(string message) : base(message)

{

}

}

[Serializable]

internal class NoMoney : Exception

{

public NoMoney()

{

}

public NoMoney(string message) : base(message)

{

}

}

}

static void Main(string[] args)

{

try

{

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

for (int i = 0; i < 10; i++)

{

wallet.Add(new Bill(5));

}

for (int i = 0; i < 50; i++)

{

wallet.Add(new Bill(10));

}

for (int i = 0; i < 10; i++)

{

wallet.Add(new Bill(20));

}

for (int i = 0; i < 20; i++)

{

wallet.Add(new Bill(100));

}

for (int i = 0; i < 111; i++) // for (int i = 0; i < 112; i++)

{

wallet.Add(new Bill(100));

}

//var binFormatted = new BinaryFormatter();

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

//{

// binFormatted.Serialize(file, wallet);

//}

var jsonFormatted = new DataContractJsonSerializer(typeof(Wallet<Bill>));

using (var file = new FileStream("file.json", FileMode.Create))

{

jsonFormatted.WriteObject(file, wallet);

}

Console.WriteLine("Количество: ");

wallet.Count();

wallet.Remove();

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

wallet.Count();

Console.WriteLine(wallet.ToString());

}

catch (Exception err)

{

Console.WriteLine(err.Message);

}

}

}

}

namespace Time

{

[Serializable]

class Time : IComparable<Time>

{

private int hours;

private int minutes;

public int seconds;

public int Hours

{

get

{

return hours;

}

set

{

if (value>=0 || value<=23)

{

hours = value;

}

else Console.WriteLine("ERROR!!!");

}

}

public int Minutes

{

get

{

return minutes;

}

set

{

if (value >= 0 || value <= 60)

{

minutes = value;

}

else Console.WriteLine("ERROR!!!");

}

}

public int Seconds

{

get

{

return seconds;

}

set

{

if (value < 0 || value > 60)

throw new SecondsException("Неправильно заданы секунды");

else

seconds = value;

}

}

static Time() // статический конструктор (закрыт, без параметров, нельзя вызвать явно)

{

}

public Time(int hours, int minutes, int seconds)

{

this.Hours = hours;

this.Minutes = minutes;

this.Seconds = seconds;

}

public int CompareTo(Time time)

{

if ((this.Hours > time.Hours) || (this.Hours == time.Hours && this.Minutes > time.Minutes))

{

return 1;

}

else if ((this.Hours < time.Hours) || (this.Hours == time.Hours && this.Minutes < time.Minutes))

{

return -1;

}

else return 0;

}

public void ShowClassInfo()

{

Console.WriteLine($"Час: {this.Hours}");

Console.WriteLine($"Минуты: {this.Minutes}");

Console.WriteLine($"Секунды: {this.Seconds}");

}

public override string ToString()

{

return ($"\t{this.Hours}:{this.Minutes}:{this.Seconds}");

}

}

[Serializable]

class SecondsException : Exception

{

public SecondsException(string message) : base(message)

{

Console.WriteLine("Исправить секунды.");

}

}

class Program

{

static void Main(string[] args)

{

try

{

Time[] time = new Time[5];

time[0] = new Time(23, 45, 34);

time[1] = new Time(4, 10, 20);

time[2] = new Time(1, 20, 55);

time[3] = new Time(9, 56, 23);

time[4] = new Time(10, 45, 34);

foreach (var n in time)

{

n.ShowClassInfo();

Console.WriteLine();

}

var night = from t in time

where t.Hours >= 0 && t.Hours <= 5

select t;

foreach (Time t in night)

{

Console.WriteLine($"Ночь: {t.ToString()}");

}

var moning = from u in time

where u.Hours >= 6 && u.Hours <= 12

select u;

foreach (Time u in moning)

{

Console.WriteLine($"Утро: {u.ToString()}");

}

var day = from d in time

where d.Hours >= 13 && d.Hours <= 24

select d;

foreach (Time d in day)

{

Console.WriteLine($"День: {d.ToString()}");

}

// запись в файл

string writePath = @"E:\БГТУ (Теория, лабы, подготовка к экзам)\2 курс\1 семестр\Экзамены\ООП\trening\Time (easy)\Time (easy)\time.txt";

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

{

foreach (Time d in day)

{

sw.WriteLine($"День: {d.ToString()}");

}

foreach (Time t in night)

{

sw.WriteLine($"Ночь: {t.ToString()}");

}

foreach (Time u in moning)

{

sw.WriteLine($"Утро: {u.ToString()}");

}

}

var jsonFormatted = new DataContractJsonSerializer(typeof(Time[]));

using (var file = new FileStream("file.json", FileMode.Create))

{

jsonFormatted.WriteObject(file, time);

}

Console.ReadKey();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

namespace UserList\_Dictionary

{

public interface IUseable

{

void Add(int key, User value);

bool Delete(int key);

void Find(int key);

}

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 void Print()

{

Console.WriteLine($"Имя абонента: {this.Name}, Номер: {this.Number}, Тариф: {this.Tarif}\n");

}

}

public class UserList : IUseable

{

public Dictionary<int, User> list;

public UserList()

{

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

}

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 ");

}

public void Print()

{

foreach(var i in list)

Console.WriteLine($"Ключ: {i.Key}, Имя: {i.Value.Name}, Номер: {i.Value.Number}");

}

}

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();

//4

// Подсчитать сколько телефонов заканчиваются на цифру 0 или 1:

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

Console.WriteLine(num);

//5

// Сариализация и десериализация в бинарный формат

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}");

}

}

catch (UserNotFound ex)

{

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

}

}

}

}

namespace EXAMS (6-2)

{

public class School

{

public event Action Schoolarhsip;

public void Money()

{

Schoolarhsip?.Invoke();

}

}

public interface IPay

{

public void Pay(int sum);

}

public class PupulCard : IPay

{

public int balance;

public class ExpiredDate

{

public int Mounth;

public int Year;

}

public ExpiredDate Date;

public int number;

public enum lockStatus

{ locked = 0, unlocked = 1 };

public lockStatus status = lockStatus.locked;

public PupulCard(int bal, ExpiredDate date, int num, lockStatus stat)

{

balance = bal;

Date = date;

number = num;

status = stat;

}

public void GetMoney()

{

balance += 100;

if (Date.Year + 2000 != DateTime.Now.Year)

{

status = lockStatus.locked;

}

}

public void Pay(int sum)

{

if (status == lockStatus.locked)

{ Console.WriteLine("No auth"); return; }

if (balance + 10 < sum)

{

Console.WriteLine("Not enough");

}

else

{

balance -= sum;

if (balance < 0)

{

using (var file = new StreamWriter("balance.txt", true))

{

file.WriteLine($"{DateTime.Now} {number} {-balance}");

}

}

}

}

}

class Program

{

static void Main(string[] args)

{

var card = new PupulCard(10, new PupulCard.ExpiredDate { Mounth = 10, Year = 10 }, 10, PupulCard.lockStatus.unlocked);

var card2 = new PupulCard(12, new PupulCard.ExpiredDate { Mounth = 10, Year = 22 }, 11, PupulCard.lockStatus.unlocked);

var card3 = new PupulCard(15, new PupulCard.ExpiredDate { Mounth = 01, Year = 22 }, 12, PupulCard.lockStatus.unlocked);

card3.Pay(40);

var sc = new School();

sc.Schoolarhsip += card.GetMoney;

sc.Schoolarhsip += card2.GetMoney;

sc.Money();

var col = new List<PupulCard> { card, card2, card3 };

var selected = col.Where(x => x.status == PupulCard.lockStatus.locked).OrderBy(x => x.balance).First();

var selected2 = col.Where(x => x.status == PupulCard.lockStatus.unlocked).Count();

Console.WriteLine(selected.number);

Console.WriteLine();

Console.WriteLine(selected2);

}

}

}

namespace EXAMS (6-3)

{

[Serializable]

public class Shtime

{

private int hours;

private int mins;

public Shtime(int h, int m)

{

Hours = h;

Minutes = m;

}

public Shtime()

{

Hours = 0;

Minutes = 0;

}

public int Hours

{

get { return hours; }

set

{

if (value < 0 || value > 24)

{

throw new ArgumentException("bruh this hours wack ");

}

else

hours = value;

}

}

public int Minutes

{

get { return mins; }

set

{

if (value < 0 || value > 60)

throw new ArgumentException("bruh this minutes wack too");

else

mins = value;

}

}

public static Shtime operator ++(Shtime one)

{

if (one.mins >= 15)

{

one.Hours++;

one.Minutes = (one.Minutes + 45) - 60;

}

else

{

one.Minutes += 45;

}

return one;

}

public static Shtime operator --(Shtime one)

{

if (one.Minutes <= 45)

{

one.Hours--;

one.Minutes = (one.Minutes - 45) + 60;

}

else

{

one.Minutes -= 45;

}

return one;

}

public void Study()

{

Hours = 8;

Minutes = 0;

}

}

public class Study

{

public event Action IsTimeToStudy;

public void DoStudy()

{

IsTimeToStudy?.Invoke();

}

}

class Program

{

static void Main(string[] args)

{

var time = new Shtime(12, 00);

var time\_2 = new Shtime(15, 00);

time++;

time++;

Console.WriteLine(time.Hours);

Console.WriteLine(time.Minutes);

var xml = new XmlSerializer(typeof(Shtime));

var json = new DataContractJsonSerializer(typeof(Shtime));

using (var file = new FileStream("bruh.xml", FileMode.Create))

{

xml.Serialize(file, time);

}

using (var file = new FileStream("bruh.json", FileMode.Create))

{

json.WriteObject(file, time);

}

using (var file = new FileStream("bruh.xml", FileMode.Open))

{

Shtime time2 = (Shtime)xml.Deserialize(file);

}

using (var file = new FileStream("bruh.json", FileMode.Open))

{

Shtime time3 = (Shtime)json.ReadObject(file);

}

var Stud = new Study();

Stud.IsTimeToStudy += time.Study;

Stud.IsTimeToStudy += time\_2.Study;

Stud.DoStudy();

Console.WriteLine($"{time.Hours}:{time.Minutes}");

Console.WriteLine($"{time\_2.Hours}:{time\_2.Minutes}");

var time\_3 = new Shtime(15, 0);

Console.WriteLine();

var col = new List<Shtime>

{

time\_3, time,time\_2,

};

col = col.Where(x => x.Minutes == 0).OrderBy(x => x.Hours).ToList();

col.ForEach(x => Console.WriteLine($"{x.Hours}:{x.Minutes}"));

}

}

}

}

namespace EXAMS (6-41)

{

public interface IEdit

{

public void Delete()

{

}

}

public abstract class Redactor

{

public void Delete()

{

}

public StringBuilder text;

}

public class Document : Redactor, IEdit

{

public Document(string words)

{

text = new StringBuilder(words);

}

public new void Delete()

{

var buff = text.ToString().Split(" ").Where(x => x != "");

text.Clear();

text.Append(String.Join(" ", buff));

}

void IEdit.Delete()

{

Delete();

var buff = text.ToString().Split(' ').First();

text.Clear();

text.Append(buff);

}

public void print()

{

using (var file = new StreamWriter($"{DateTime.Now.ToString("HH-mm")}.txt", true))

{

file.WriteLine(text.ToString());

}

}

public override string ToString()

{

return text.ToString();

}

public override int GetHashCode()

{

return new Random().Next();

}

}

public class Book : Document

{

public Book(string words) : base(words)

{

}

public new void print()

{

using (var file = new StreamWriter($"{DateTime.Now.ToString("HH-mm")}.txt", true))

{

foreach (var i in text.ToString().Split(".").ToList())

{

file.WriteLine(i);

}

}

}

}

public static class BookExtensions

{

public static void ToBeContinue(this Book b)

{

b.text.Append("...");

}

}

class Program

{

static void Main(string[] args)

{

var doc = new Document(" BRUH 123 HELLO ");

doc.Delete();

Console.WriteLine(doc.text);

((IEdit)doc).Delete();

Console.WriteLine(doc.text);

Console.WriteLine(doc.GetHashCode());

var bok = new Book("THIS is a text.Very Important.Thanks.for reading.");

bok.ToBeContinue();

bok.print();

Console.WriteLine();

var Archive = new List<Document> { doc, new Document(" text number two"), new Book(" text number 3.Yes"), new Document("I'm tired now bruh"), new Book("BRHU .NDASDA.ASDASDASDA.") };

Archive.ForEach(x => { x.Delete(); x.print(); });

}

}

namespace ConsoleApp3

{

public class Card: IPay

{

private int balance;

public class ExDate

{

public int balance1;

public int monght;

public int Monght

{

get

{

return monght;

}

set

{

monght = value;

}

}

public int yers;

public int Yehrs

{

get

{

return yers;

}

set

{

yers = value;

}

}

public int number;

//public void Input(int a)

//{

// if (a == balance1)

// {

// Console.WriteLine(number);

// }

//}

}

void IPay.Pay(int a, ExDate exDate)

{

if (a <= -100)

{

throw new Exception("Error");

}

balance = a;

exDate.balance1 = balance;

exDate.number = balance + 1000;

}

}

public interface IPay

{

void Pay(int a, Card.ExDate exDate);

}

class Program

{

static void Main(string[] args)

{

Card.ExDate card = new Card.ExDate();

Card.ExDate card1 = new Card.ExDate();

Card.ExDate card2 = new Card.ExDate();

var rdf = new Card();

((IPay)rdf).Pay(10, card);

((IPay)rdf).Pay(-19, card1);

((IPay)rdf).Pay(90, card2);

Card.ExDate[] arr = { card, card1, card2 };

var lin = (from i in arr

select i.balance1).Max();

Console.WriteLine(lin);

foreach(var i in arr)

{

if(i.balance1 == lin)

{

Console.WriteLine("Номер карты с балансом {0}: {1} ",lin,i.number);

}

}

}

}

}

**СЕРИАЛИЗАЦИЯ ВО ВСЕ ФОРМАТЫ:**

var binFormatted = new BinaryFormatter();

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

{

binFormatted.Serialize(file, users);

}

var soapFormatted = new SoapFormatter();

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

{

soapFormatted.Serialize(file, users);

}

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

var XMLFormatted = new XmlSerializer(typeof(WedNet<User>));

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

{

XMLFormatted.Serialize(file, users);

Console.WriteLine("Объект сериализован");

}

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

var jsonFormatted = new DataContractJsonSerializer(typeof(List<User>));

using (var file = new FileStream("file.json", FileMode.Create))

{

jsonFormatted.WriteObject(file, users);

}

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

{

var list = jsonFormatted.ReadObject(file) as List<User>;

foreach (User i in list)

{

Console.WriteLine(i);

}

}

**Операции со строками:**

text = text.Replace("хороший", "плохой");

// индекс последнего символа

int ind = text.Length - 1;

// вырезаем последний символ

text = text.Remove(ind);

// вставка  
text = text.Insert(8, subString);

// Разделение  
string[] words = text.Split(new char[] { ' ' });

// Поиск  
string s1 = "hello world";

char ch = 'o';

int indexOfChar = s1.IndexOf(ch); // равно 4

// сравнение  
int result = String.Compare(s1, s2);  
//соединение  
string s4 = String.Concat(s3, "!!!"); // результат: строка "hello world!!!"

**Перегрузка операторов:**

public static bool operator >(Variant23 first, Variant23 second )

{

return first.length > second.length;

}

**Запись в файл:**

string writePath = @"E:\БГТУ (Теория, лабы, подготовка к экзам)\2 курс\1 семестр\Экзамены\ООП\trening\Time (easy)\Time (easy)\time.txt";

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

{

foreach (Time d in day)

{

sw.WriteLine($"День: {d.ToString()}");

}

}

**Массивы:**

// Объявляем двумерный массив

int[,] myArr = new int[4, 5];

Random ran = new Random();

// Инициализируем данный массив

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 5; j++)

{

myArr[i, j] = ran.Next(1, 15);

Console.Write("{0}\t", myArr[i, j]);

}

Console.WriteLine();

}

// Объявляем одномерный массив

string[] str = new string[] { "GH", "JHDU", "GFDk", "GYGSk" };

foreach (string i in str)

Console.Write(i + " ");

Console.WriteLine();

double[] dbl = { 13.2, 235.64, 56.6 };

foreach (double x in dbl)

Console.Write(x + " ");

Console.WriteLine();

// Объявляем зубчатый массив

int[][] arr = new int[2][];

arr[0] = new int[] { 2, 3, 5, 6 };

arr[1] = new int[] { 9, 8, 7, 6, 5, 4 };

foreach (int[] x in arr)

{

foreach (int mass in x)

{

Console.Write($"{mass} \t");

}

Console.WriteLine();

}

Console.WriteLine();

int[][] numbers = new int[3][];

numbers[0] = new int[] { 1, 2 };

numbers[1] = new int[] { 1, 2, 3 };

numbers[2] = new int[] { 1, 2, 3, 4, 5 };

foreach (int[] row in numbers)

{

foreach (int number in row)

{

Console.Write($"{number} \t");

}

Console.WriteLine();

}

**LINQ**

Select: определяет проекцию выбранных значений

Where: определяет фильтр выборки

OrderBy: упорядочивает элементы по возрастанию

OrderByDescending: упорядочивает элементы по убыванию

ThenBy: задает дополнительные критерии для упорядочивания элементов возрастанию

ThenByDescending: задает дополнительные критерии для упорядочивания элементов по убыванию

Join: соединяет две коллекции по определенному признаку

GroupBy: группирует элементы по ключу

ToLookup: группирует элементы по ключу, при этом все элементы добавляются в словарь

GroupJoin: выполняет одновременно соединение коллекций и группировку элементов по ключу

Reverse: располагает элементы в обратном порядке

All: определяет, все ли элементы коллекции удовлятворяют определенному условию

Any: определяет, удовлетворяет хотя бы один элемент коллекции определенному условию

Contains: определяет, содержит ли коллекция определенный элемент

Distinct: удаляет дублирующиеся элементы из коллекции

Except: возвращает разность двух коллекцию, то есть те элементы, которые создаются только в одной коллекции

Union: объединяет две однородные коллекции

Intersect: возвращает пересечение двух коллекций, то есть те элементы, которые встречаются в обоих коллекциях

Count: подсчитывает количество элементов коллекции, которые удовлетворяют определенному условию

Sum: подсчитывает сумму числовых значений в коллекции

Average: подсчитывает cреднее значение числовых значений в коллекции

Min: находит минимальное значение

Max: находит максимальное значение

Take: выбирает определенное количество элементов

Skip: пропускает определенное количество элементов

TakeWhile: возвращает цепочку элементов последовательности, до тех пор, пока условие истинно

SkipWhile: пропускает элементы в последовательности, пока они удовлетворяют заданному условию, и затем возвращает оставшиеся элементы

Concat: объединяет две коллекции

Zip: объединяет две коллекции в соответствии с определенным условием

First: выбирает первый элемент коллекции

FirstOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

Single: выбирает единственный элемент коллекции, если коллекция содердит больше или меньше одного элемента, то генерируется исключение

SingleOrDefault: выбирает первый элемент коллекции или возвращает значение по умолчанию

ElementAt: выбирает элемент последовательности по определенному индексу

ElementAtOrDefault: выбирает элемент коллекции по определенному индексу или возвращает значение по умолчанию, если индекс вне допустимого диапазона

Last: выбирает последний элемент коллекции.