using System.Runtime.Serialization;

using System.Runtime.Serialization.Json;

namespace \_1\_1

{

interface Figure

{

void Print();

}

[DataContract]

class Rectangle : Figure

{

[DataMember]

public int X { get; set; }

[DataMember]

public int Y { get; set; }

[DataMember]

public int H { get; set; }

[DataMember]

public int L { get; set; }

[DataMember]

public string Color { get; set; }

public Rectangle()

{

X = 0;

Y = 0;

H = 0;

L = 0;

Color = "Белый";

}

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

{

X = x;

Y = y;

Color = color;

}

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

{

L = l;

H = h;

}

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

{

A.H += i;

A.L += i;

return A;

}

public int Sqr()

{

return H \* L;

}

public virtual void Print()

{

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

}

public override string ToString()

{

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

}

}

[DataContract]

class SerializableList

{

[DataMember]

public List<Rectangle> Rectangles { get; set; }

public SerializableList(List<Rectangle> rectangles)

{

Rectangles = rectangles;

}

}

[DataContract]

class Program

{

static void Main(string[] args)

{

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

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

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

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

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

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

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

listochek.Add(rec1);

listochek.Add(rec2);

listochek.Add(rec3);

listochek.Add(rec4);

listochek.Add(rec5);

listochek.Add(rec6);

rec2 += 20;

rec1.Print();

var sortedRectangles = listochek.OrderBy(r => r.X).ThenBy(r => r.Y).ThenBy(r => r.Sqr()).ToList();

sortedRectangles.First().Print();

sortedRectangles.Last().Print();

DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(SerializableList));

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

{

SerializableList serializableList = new SerializableList(listochek);

jsonFormatter.WriteObject(fs, serializableList);

}

}

}

}

-------------------------------------------------------------------------------------------------

namespace \_1\_2

{

abstract class Transport

{

public string Name { get; set; }

public Transport (string name)

{

this.Name = name;

}

}

public interface IAirable

{

void Check();

void Fly();

}

public interface IAir

{

void Check();

}

public enum Status\_enum

{

Fly,

Ready,

Stop

}

class Air : Transport, IAirable, IAir

{

public int CountOfPassengers { get; set; }

public int Speed { get; set; }

public Status\_enum Status { get; set; }

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

public void Check()

{

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

{

Status = Status\_enum.Stop;

}

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

{

Status = Status\_enum.Ready;

}

else if (Status == Status\_enum.Ready && CountOfPassengers > 0 && Speed > 0)

{

Status = Status\_enum.Fly;

}

}

void IAir.Check()

{

if (CountOfPassengers > 20 && CountOfPassengers < 100)

{

Console.WriteLine("Ready");

}

}

public void Fly()

{

if (Status == Status\_enum.Fly)

{

Console.WriteLine("Flying");

}

else

{

throw new Exception("Cannot fly. Check the status.");

}

}

}

class Program

{

static void Main(string[] args)

{

using (StreamWriter sw = new StreamWriter("text.txt"))

{

Air air = new Air("Airplane");

air.CountOfPassengers = 10;

air.Speed = 0;

air.Check();

((IAir)air).Check();

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

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

air.Speed = 100;

air.Check();

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

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

air.Fly();

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

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

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

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

airList?.Add(new Air("Airplane4"));

airList?.Add(new Air("Airplane5"));

airList[0].CountOfPassengers = 10;

airList[0].Speed = 200;

airList[0].Check();

airList[1].CountOfPassengers = 10;

airList[1].Speed = 0;

airList[1].Check();

airList[2].CountOfPassengers = 0;

airList[2].Speed = 0;

airList[2].Check();

airList[3].CountOfPassengers = 30;

airList[3].Speed = 300;

airList[3].Check();

airList[4].CountOfPassengers = 25;

airList[4].Speed = 250;

airList[4].Check();

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

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

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

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

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

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

}

}

}

}

-------------------------------------------------------------------------------------------------

using System.Runtime.Serialization.Json;

using System.Runtime.Serialization;

using System.Collections;

namespace \_2\_6

{

public enum Status

{

Signin,

Signout

}

[DataContract]

class User: IComparable

{

[DataMember]

private string email;

[DataMember]

public string password;

[DataMember]

public Status status;

public override string ToString()

{

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

}

public override bool Equals(object? obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

User other = (User)obj;

return this.email == other.email;

}

public override int GetHashCode()

{

return email.GetHashCode();

}

public int CompareTo(object? obj)

{

if (obj == null)

return 1;

User other = (User)obj;

if (other == null)

return 1;

else

return email.CompareTo(other.email);

}

/\* public int CompareTo(object? obj)

{

if (obj == null)

return 1;

User other = (User)obj;

if (other == null)

return 1;

else

{

int emailComp = email.CompareTo(other.email);

if (emailComp != 0)

{

return emailComp;

}

else

{

return password.CompareTo(other.password);

}

}

}\*/

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

{

this.email = email;

this.password = password;

this.status = status;

}

}

[DataContract]

class WebNet

{

[DataMember]

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

public void AddUser(User user)

{

users.AddLast(user);

}

public void RemoveUser(User user)

{

users.Remove(user);

}

public void PrintUsers()

{

Console.WriteLine("Users:");

foreach (User user in users)

{

Console.WriteLine(user);

}

}

}

class Program

{

static void Main(string[] args)

{

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

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

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

Console.WriteLine(user1);

Console.WriteLine(user2);

Console.WriteLine(user3);

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

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

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

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

WebNet github = new WebNet();

github.AddUser(user1);

github.AddUser(user2);

github.AddUser(user3);

github.PrintUsers();

var users = github.users.Where(user => user.status == Status.Signin);

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

foreach (User user in users)

{

Console.WriteLine(user);

}

/\* var usersS = github.users.Where(u => u.password.Length < 8 && u.password.All(c => c >= '0' && c <= '9'));

Console.WriteLine("Users with password:");

foreach (User user in usersS)

{

Console.WriteLine(user);

}\*/

DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(WebNet));

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

{

jsonFormatter.WriteObject(fs, github);

}

}

}

}---------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3\_2

{

public enum Status

{

free,

busy

}

class Location

{

public int lat { get; set; }

public int lon { get; set; }

public int speed { get; set; }

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

{

this.lat = lat;

this.lon = lon;

this.speed = speed;

}

}

class Taxi

{

public string Number { get; set; }

public Location Location { get; set; }

public Status Status;

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

{

this.Number = number;

this.Location = location;

this.Status = status;

}

public override string ToString()

{

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

}

}

class Park<T>

{

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

public void Add(T item)

{

list.Add(item);

}

public void Remove(T item)

{

list.Remove(item);

}

public void ClearPark()

{

list.Clear();

}

public T Find(Func<T, bool> predicate)

{

return list.FirstOrDefault(predicate);

}

public override string ToString()

{

string str = "";

foreach (T item in list)

{

str += item + "\n";

}

return str;

}

}

class Program

{

static void Main(string[] args)

{

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

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

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

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

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

Console.WriteLine("Список такси:");

Console.WriteLine(uber.ToString());

Console.WriteLine("Введите вашу текущую координату (широта):");

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

Console.WriteLine("Введите вашу текущую координату (долгота):");

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

uber.list.Sort((x, y) => CalculateDistance(x.Location.lat, x.Location.lon, lat, lon).CompareTo(CalculateDistance(y.Location.lat, y.Location.lon, lat, lon)));

Console.WriteLine(uber.ToString());

var taxi = uber.list.OrderBy(x => CalculateDistance(x.Location.lat, x.Location.lon, lat, lon)).First();

Console.WriteLine("Ближайшее такси:");

Console.WriteLine(taxi);

using (StreamWriter fs = new StreamWriter("txt.txt"))

{

fs.WriteLine(taxi);

};

static double CalculateDistance(int lat1, int long1, int lat2, int long2)

{

return Math.Sqrt(Math.Pow(lat2 - lat1, 2) + Math.Pow(long2 - long1, 2));

}

}

}

}

-------------------------------------------------------------------------------------------------

using System.Collections;

namespace \_3\_3

{

public class SomeString: IComparer

{

public string str;

public SomeString(string str)

{

this.str = str;

}

public override bool Equals(object? obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

SomeString other = (SomeString)obj;

return (this.str.Length == other.str.Length && this.str.First() == other.str.First() && this.str.Last() == other.str.Last());

}

public int Compare(object s1, object s2)

{

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

return 1;

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

return -1;

else return 0;

}

public static SomeString operator +(SomeString s1, char s2)

{

return new SomeString(s1.str + s2);

}

public static SomeString operator -(SomeString s1, char s2)

{

try

{

if (s1.str.Length == 0)

throw new Exception("Строка пустая");

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

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

}

}

public static class SomeStringExtention

{

public static int CountSpaces(this SomeString s1)

{

int count = 0;

foreach (var s in s1.str)

{

if (s == ' ')

{

count++;

}

}

return count;

}

public static string Remove(this SomeString s1)

{

var charsToRemove = new string[] { ".", ",", "!", ";", "-" };

foreach (var s in charsToRemove)

{

s1.str = s1.str.Replace(s, "");

}

return s1.str;

}

}

class Program

{

static void Main(string[] args)

{

using (StreamWriter sw = new StreamWriter("text.txt"))

{

SomeString s1 = new SomeString("мама папа я,,!");

SomeString s2 = new SomeString("мама папа я,,!");

SomeString s3 = new SomeString("мама я!");

sw.WriteLine(s1.Equals(s2));

sw.WriteLine(s2.Equals(s3));

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

s1 += 'a';

s2 -= ' ';

sw.WriteLine(s1.str);

sw.WriteLine(s2.str);

sw.WriteLine(s3.str);

sw.WriteLine(SomeStringExtention.CountSpaces(s3));

sw.WriteLine(SomeStringExtention.Remove(s1));

sw.WriteLine(s2.str);

sw.WriteLine(s3.str);

SomeString[] someStrings = new SomeString[3];

someStrings[0] = s1;

someStrings[1] = s2;

someStrings[2] = s3;

var someString = someStrings.Sum(s => s.CountSpaces());

sw.Write(someString);

}

}

}

}

using System.Collections;

namespace \_3\_21

{

class Program

{

interface ICleanable

{

void Clean();

}

public enum Spec

{

poit,

isit,

mobile

}

class Stud

{

public string? Name { get; set; }

public int Group { get; set; }

public int Course { get; set; }

public Spec specialization;

public int Exam1, Exam2, Exam3;

public Stud(string name, int group, int course, Spec spec, int mark1, int mark2, int mark3)

{

Name = name;

Group = group;

Course = course;

specialization = spec;

Exam1 = mark1;

Exam2 = mark2;

Exam3 = mark3;

}

public (int, int, double) Marks()

{

int[] arr = new int[] { Exam1, Exam2, Exam3 };

int max = arr.Max();

int min = arr.Min();

double avg = Math.Round(arr.Average(), 2);

return (max, min, avg);

}

}

class Group : ICleanable

{

List<Stud> groups = new List<Stud>();

public List<Stud> GetStudents()

{

return groups;

}

public void Add(Stud student)

{

groups.Add(student);

}

public void Print()

{

foreach (Stud student in groups)

{

Console.WriteLine($"{student.Name} {student.Course} {student.Group}");

}

}

public void Clean()

{

groups.Clear();

}

}

static void Main(string[] args)

{

Stud student1 = new Stud("Sveta", 2, 2, Spec.isit, 9, 9, 5);

Stud student2 = new Stud("Lera", 2, 2, Spec.isit, 9, 7, 10);

Stud student3 = new Stud("Sneg", 2, 2, Spec.isit, 9, 7, 10);

Stud student4 = new Stud("Liza", 2, 2, Spec.mobile, 5, 10, 10);

Stud student5 = new Stud("Anya", 2, 2, Spec.mobile, 5, 10, 8);

Console.WriteLine(student1.specialization);

Console.WriteLine(student1.Marks());

Group group = new Group();

group.Add(student1);

group.Add(student2);

group.Add(student3);

group.Add(student4);

group.Add(student5);

group.Print();

var maxAvgBySpecialization = group.GetStudents().GroupBy(s => s.specialization).Select(g => g.OrderByDescending(s => s.Marks().Item3).First());

Console.WriteLine("\nСтуденты, набравшие максимальный средний балл по каждой специализации:");

foreach (var stud in maxAvgBySpecialization)

{

Console.WriteLine($"{stud.Name} - {stud.specialization} - {stud.Marks().Item3}");

}

try

{

group.Clean();

group.Print();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

-------------------------------------------------------------------------------------------------------

using System.Reflection;

using System.Reflection.Metadata.Ecma335;

namespace \_3\_3

{

class Program

{

class DeleteException : Exception

{

public DeleteException(string error) : base(error) { }

}

class D2Point

{

public int a;

public int X { get; set; }

public int Y { get; set; }

public D2Point(int x, int y)

{

X = x;

Y = y;

}

public void ChangeZnak()

{

X = -X; Y = -Y;

}

public static bool operator ==(D2Point a, D2Point b)

{

if (Equals(a, b))

return true;

if (a is null || b is null)

return false;

return a.X == b.X && a.Y == b.Y;

}

public static bool operator !=(D2Point a, D2Point b)

{

return !(a == b);

}

public override string ToString()

{

return "X = " + X + "Y = " + Y;

}

}

class D2Path

{

public delegate void StateHandler();

public event StateHandler? Change;

public void ChangeEwe()

{

Change?.Invoke();

}

List<D2Point> points = new List<D2Point>();

public void Add(D2Point item)

{

points.Add(item);

}

public void Delete(int item)

{

if (points.Count > 0) points.RemoveAt(item);

else throw new DeleteException("Попытка удаления из пустой коллекции.");

}

public void Clear()

{

points.Clear();

}

public (int, int, int, int) CountPoints()

{

int fq = 0, sq = 0, tq = 0, foq = 0;

foreach (var x in points)

{

if (x.X > 0 && x.Y > 0) fq++;

if (x.X < 0 && x.Y > 0) sq++;

if (x.X < 0 && x.Y < 0) tq++;

if (x.X > 0 && x.Y < 0) foq++;

}

return (fq, sq, tq, foq);

}

public override string ToString()

{

string result = "";

foreach (var point in points)

{

result += $"Point: X = {point.X}, Y = {point.Y}\n";

}

return result;

}

}

static void Main(string[] args)

{

D2Point p1 = new D2Point(3, 5);

D2Point p2 = new D2Point(4, 7);

D2Point p3 = new D2Point(4, 7);

D2Path d2Path = new D2Path();

d2Path.Add(p1);

d2Path.Add(p2);

d2Path.Add(p3);

d2Path.Change += p1.ChangeZnak;

d2Path.ChangeEwe();

Console.WriteLine(d2Path.ToString());

Console.WriteLine(p2 == p3);

Console.WriteLine(p1 == p3);

try

{

d2Path.Delete(0);

}

catch (DeleteException ex)

{

Console.WriteLine(ex.Message);

}

Console.WriteLine(d2Path.CountPoints());

d2Path.Clear();

Type myType = typeof(D2Path);

//Console.WriteLine($"Name: {myType.Name}"); // получаем краткое имя типа

//Console.WriteLine($"Full Name: {myType.FullName}"); // получаем полное имя типа

//Console.WriteLine($"Namespace: {myType.Namespace}"); // получаем пространство имен типа

//Console.WriteLine($"Is struct: {myType.IsValueType}"); // является ли тип структурой

//Console.WriteLine($"Is class: {myType.IsClass}"); // является ли тип классом

//Console.WriteLine("Реализованные интерфейсы:");

//foreach (Type i in myType.GetInterfaces())

//{

// Console.WriteLine(i.Name);

//}

//foreach (MemberInfo member in myType.GetMembers(BindingFlags.DeclaredOnly | BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public))

//{

// Console.WriteLine($"{member.DeclaringType} {member.MemberType} {member.Name}");

//}

//foreach (MethodInfo method in myType.GetMethods())

//{

// string modificator = "";

// // если метод статический

// if (method.IsStatic) modificator += "static ";

// // если метод виртуальный

// if (method.IsVirtual) modificator += "virtual ";

// Console.WriteLine($"{modificator}{method.ReturnType.Name} {method.Name} ()");

//}

foreach (ConstructorInfo member in myType.GetConstructors())

{

Console.WriteLine($"{member.DeclaringType} {member.MemberType} {member.Name}");

}

foreach (FieldInfo member in myType.GetFields())

{

Console.WriteLine($"{member.DeclaringType} {member.MemberType} {member.Name}");

}

}

}

}

namespace \_7\_3

{

class Program

{

class Button

{

public string? caption;

public double w, h;

public Button(string caption, int width, int height)

{

this.caption = caption;

w = width;

h = height;

}

}

class CheckButton : Button

{

public CheckButton(string caption, int width, int height, State state) : base(caption, width, height)

{

this.state = state;

}

State state = State.Checked;

public enum State

{

Checked,

Unchecked

}

public override string? ToString()

{

return caption;

}

public override bool Equals(object? obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Button? btn = obj as Button;

return ((btn?.caption == caption) && (btn?.h == h) && (btn?.w == w));

}

public void Check()

{

if (state == State.Checked) {

state = State.Unchecked;

Console.WriteLine($"Состояние: {state}");

}

else {

state = State.Checked;

Console.WriteLine($"Состояние: {state}");

}

}

public void Zoom(int percent)

{

Console.WriteLine($"Исходные размеры h: {h} w: {w}");

w = w \* (1 - percent / 100.0);

h = h \* (1 - percent / 100.0);

Console.WriteLine($"Измененные размеры h: {h} w: {w}");

}

}

class User

{

public delegate void StateHandler2();

public event StateHandler2? Click;

public delegate void StateHandler(int percent);

public event StateHandler? Resize;

public void OnClick()

{

Console.WriteLine("Вызван метод Click");

Click?.Invoke();

}

public void OnResize()

{

Console.WriteLine("Вызван метод Resize");

Resize?.Invoke(20);

}

}

static void Main(string[] args)

{

CheckButton btt1 = new CheckButton("Button", 3, 4, CheckButton.State.Checked);

CheckButton btt2 = new CheckButton("Button", 3, 4, CheckButton.State.Unchecked);

if (btt1.Equals(btt2))

{

Console.WriteLine("Равны");

}

else

{

Console.WriteLine("Не равны");

}

btt1.Check();

btt1.Check();

btt1.Check();

btt1.Check();

btt1.Check();

Console.WriteLine("Уменьшение на заданный процент: ");

btt1.Zoom(50);

User user = new User();

user.Click += btt1.Check;

user.Resize += btt2.Zoom;

user.OnClick();

user.OnResize();

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

Button button1 = new Button("Button1", 20, 30);

Button button2 = new Button("Button2", 30, 20);

Button button3 = new Button("Button3", 20, 20);

Button button4 = new Button("Button4", 10, 60);

Button button5 = new Button("Button5", 2, 30);

Button button6 = new Button("Button6", 30, 30);

CheckButton checkbutton1 = new CheckButton("CheckButton1", 100, 200, CheckButton.State.Checked);

buttons.AddLast(button1);

buttons.AddLast(button2);

buttons.AddLast(button3);

buttons.AddLast(button4);

buttons.AddLast(button5);

buttons.AddLast(button6);

buttons.AddLast(checkbutton1);

var s = buttons.Where(x => (x.h \* x.w == 600));

foreach (Button a in s)

{

Console.WriteLine($"Кнопка {a.caption} Размеры h: {a.h} w: {a.w}");

}

var s2 = buttons.Count(x => x is CheckButton);

Console.WriteLine($"Количество кнопок типа CheckButton: {s2}");

}

}

}

-----------------------------------------------------------------------------------------------

using System.Collections;

namespace \_8\_1

{

public interface IEnumerable

{

IEnumerator GetEnumerator();

}

public class Item

{

public string name { get; set; }

public int ID { get; set; }

public double price { get; set; }

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

{

this.name = name;

this.ID = ID;

this.price = price;

}

public override string ToString()

{

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

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public void OnSale()

{

price \*= 0.5;

Console.WriteLine($"Распродажа сейчас");

}

}

public class Manager

{

public event \_Sale sale;

public void Sale()

{

Console.WriteLine("Вызван метод Sale()");

sale?.Invoke();

}

}

public delegate void \_Sale();

public class Shop : IEnumerable

{

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

public void Add(Item item)

{

queue.Enqueue(item);

}

public void Remove()

{

queue.Dequeue();

}

public void Delete()

{

queue.Clear();

}

public IEnumerator GetEnumerator()

{

return queue.GetEnumerator();

}

}

class Program

{

static void Main(string[] args)

{

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

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

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

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

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

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

queue.Enqueue(item1);

queue.Enqueue(item2);

queue.Enqueue(item3);

queue.Enqueue(item4);

queue.Enqueue(item5);

Console.WriteLine("Items in the queue:");

foreach (Item item in queue)

{

Console.WriteLine(item);

}

Console.WriteLine(item1.ToString());

Console.WriteLine(item2.GetHashCode());

Manager manager = new Manager();

manager.sale += item1.OnSale;

manager.sale += item3.OnSale;

manager.Sale();

foreach (Item a in queue)

Console.WriteLine(a);

Console.WriteLine();

var query = queue.Where(x => x.name == "shirt").Sum(x => x.price);

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

}

}

}

-------------------------------------------------------------------------------------------------

namespace \_8\_2

{

internal class Program

{

abstract class Function

{

public virtual string Func() { return "\0"; }

public int X { get; set; }

public override int GetHashCode()

{

return 100 \* new Random().Next(100);

}

}

class Liner : Function

{

public int A { get; set; }

public int B { get; set; }

public Liner(int a, int b)

{

A = a;

B = b;

}

public override string Func()

{

return $"{A}x + {B}";

}

public override string ToString()

{

return "Liner";

}

}

class Sqr : Function

{

public int A { get; set; }

public int B { get; set; }

public int C { get; set; }

public Sqr(int a, int b, int c)

{

A = a;

B = b;

C = c;

}

public override string Func()

{

return $"{A}x2 + {B}x + {C}";

}

public override string ToString()

{

return "Sqr";

}

}

class ArrayFunct<T> where T : Function

{

public List<T> list;

public ArrayFunct()

{

list = new List<T>(new T[100]);

}

public T this[int index]

{

get { return list[index]; }

set { list[index] = value; }

}

}

static void Main(string[] args)

{

using (StreamWriter sw = new StreamWriter("txt.txt"))

{

Liner liner = new Liner(2, 3);

Liner liner2 = new Liner(3, 3);

Sqr sqr = new Sqr(2, 3, 4);

Sqr sqr2 = new Sqr(0, 3, 4);

ArrayFunct<Function> arrayFunct = new ArrayFunct<Function>();

arrayFunct[0] = sqr;

arrayFunct[1] = liner;

arrayFunct[2] = liner2;

arrayFunct[3] = sqr2;

var a = arrayFunct.list[0];

var b = arrayFunct.list[1];

((Sqr)a).GetHashCode();

((Liner)b).GetHashCode();

Console.WriteLine(a.Func());

Console.WriteLine(b.Func());

sw.WriteLine(a.Func());

sw.WriteLine(b.Func());

Console.WriteLine(a);

Console.WriteLine(b);

sw.WriteLine(a);

sw.WriteLine(b);

var minValueFunc = arrayFunct.list

.Where(x => x != null)

.OrderBy(x => x.X)

.ThenBy(x => (x is Liner linerObj) ? linerObj.A : ((x is Sqr sqrObj) ? sqrObj.A : 100))

.FirstOrDefault();

Console.WriteLine(minValueFunc?.Func());

sw.WriteLine(minValueFunc?.Func());

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_8\_4

{

public interface IScore

{

int Amount { get; set; }

int AddMoney();

int RemMoney();

}

abstract class Human

{

DateTime Date { get; set; }

}

class Person : Human, IScore

{

public static int countobj;

public string Name;

public string SecName;

public DateTime Date;

public int amount;

public int Amount { get; set; }

static Person()

{

countobj = 0;

}

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 += count;

return Amount;

}

public int RemMoney()

{

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

Amount -= count;

return Amount;

}

public static void CountobjToString()

{

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

}

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Person other = (Person)obj;

return this.Date == other.Date;

}

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();

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

Task<List<Person>> task1 = new Task<List<Person>>(() => belarus.Where(x => x.Date == aaa).ToList());

task1.Start();

Task<List<Person>> task2 = new Task<List<Person>>(() => alfa.Where(x => x.Date == aaa).ToList());

task2.Start();

Task<List<Person>> task3 = new Task<List<Person>>(() => central.Where(x => x.Date == aaa).ToList());

task3.Start();

foreach (var p in task1.Result)

{

Console.WriteLine(p.Name);

}

foreach (var p in task2.Result)

{

Console.WriteLine(p.Name);

}

foreach (var p in task3.Result)

{

Console.WriteLine(p.Name);

}

}

}

}

-----------------------------------------------------------------------------------------------

namespace Exam

{

interface IAction<T>

{

void Add(T obj);

void Remove(T obj);

void Clean();

void Info();

}

class NullSizeCollection : Exception

{

public NullSizeCollection(string message) : base(message)

{

Console.WriteLine("Коллекция пуста");

}

}

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

{

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

public List<T> List

{

get { return list; }

set { list = value; }

}

public void Add(T obj)

{

list.Add(obj);

}

public void Remove(T obj)

{

try

{

if (list.Count == 0)

{

throw new NullSizeCollection("Коллекция пустая");

}

else list.Remove(obj);

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

public void Clean()

{

try

{

if (list.Count == 0)

{

throw new NullSizeCollection("Коллекция пустая");

}

else list.Clear();

}

catch (NullSizeCollection ex)

{

Console.WriteLine(ex.Message);

}

}

public void Info()

{

foreach (var item in list)

{

Console.WriteLine(item);

}

}

}

public class Student

{

public string Name;

public string Subject;

public int Mark;

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

{

this.Name = name;

this.Subject = subject;

this.Mark = mark;

}

public override string ToString()

{

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

}

}

class Program

{

static void Main(string[] args)

{

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

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

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

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

examCard.Add(student1);

examCard.Add(student2);

examCard.Add(student3);

examCard.Info();

var query\_4 = examCard.List.Where(student => student.Mark >= 4);

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

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

foreach (var item in query\_4)

{

Console.WriteLine(item);

}

double average = query\_4.Average(student => student.Mark);

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

Random random = new Random();

foreach (var item in query\_4)

{

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

Console.WriteLine(item);

}

}

}

}

-------------------------------------------------------------------------------------------------namespace \_6\_1

{

class PinErrorException : Exception

{

public PinErrorException(string message) : base(message)

{

}

}

class CanNotException : Exception

{

public CanNotException (string message) : base(message)

{

}

}

interface ICreditCard

{

void Add(int obj);

void Get(int obj);

}

public class CreditCard : ICreditCard

{

public int balance;

public int number;

private readonly int pin;

private readonly int pin2;

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

{

this.balance = balance;

this.number = number;

this.pin = pin;

this.pin2 = pin2;

}

public override string ToString()

{

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

}

public void CheckBalance()

{

int pinInput = 0;

while (true)

{

if (pinInput < 3)

{

try

{

Console.WriteLine("Введите pin: ");

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

if (pin\_1 == pin)

{

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

break;

}

else

{

pinInput++;

throw new PinErrorException("Пароль введён неверно");

}

}

catch (PinErrorException ex)

{

LogException(ex, "CheckBalance", nameof(CreditCard));

}

}

else

{

Console.WriteLine("Введите pin2: ");

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

if (pin\_2 == pin2)

{

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

break;

}

else

{

throw new PinErrorException("Пароль введён неверно");

}

}

}

}

public void Add(int obj)

{

balance = this.balance + obj;

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

}

public void Get(int obj)

{

try

{

if (balance - obj >= 0)

{

balance = this.balance - obj;

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

}

else

{

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

}

}

catch (CanNotException ex)

{

LogException(ex, "Get", nameof(CreditCard));

}

}

private void LogException(Exception ex, string methodName, string className)

{

using (StreamWriter writer = new StreamWriter("txt.txt"))

{

writer.WriteLine($"Тип исключения: {ex.GetType().Name}");

writer.WriteLine($"Сообщение: {ex.Message}");

writer.WriteLine($"Время: {DateTime.Now}");

writer.WriteLine($"Метод: {methodName}");

writer.WriteLine($"Класс: {className}");

writer.WriteLine("-------------------------------");

}

}

}

class Program

{

static void Main(string[] args)

{

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

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

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

creditCard.CheckBalance();

creditCard.Add(45);

creditCard.Get(100);

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

creditCards.Add(creditCard);

creditCards.Add(creditCard2);

creditCards.Add(creditCard3);

var selectByMoney = creditCards

.Where(x => x.balance > 100 && x.balance < 200 && x.number.ToString().Contains("9"))

.Sum(x => x.balance);

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

}

}

}------------------------------------------------------------------------------------------------using System;

using System.Collections.Generic;

using System.Text;

namespace Edit

{

interface IEdit

{

void Delete();

}

public abstract class Redactor

{

public abstract void Delete();

public StringBuilder? Text { get; set; }

}

public class Document : Redactor, IEdit

{

public Document(string text)

{

this.Text = new StringBuilder(text);

}

public override void Delete()

{

string[] words = Text.ToString().Split(' ');

string firstWord = words[0];

Text.Clear();

Text.Append(firstWord);

}

void IEdit.Delete()

{

string trimmedText = System.Text.RegularExpressions.Regex.Replace(Text.ToString(), @"\s+", " ");

Text.Clear();

Text.Append(trimmedText);

}

public override string ToString()

{

return base.ToString();

}

public override bool Equals(object? obj)

{

if (obj == null || GetType() != obj.GetType())

return false;

Document other = (Document)obj;

return this.Text == other.Text;

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public virtual void Print(string filePath)

{

File.WriteAllText(filePath, Text.ToString());

}

}

public class Book : Document

{

public Book(string text) : base(text)

{

}

public override void Print(string filePath)

{

File.WriteAllText(filePath, Text.ToString());

}

}

public static class BookExtensions

{

public static void ToBeContinue(this Book book)

{

book.Text.Append("To be continued...");

}

}

class Programm

{

static void Main(string[] args)

{

Document doc = new Document("текст для задания");

((IEdit)doc).Delete();

doc.Print("doc\_output.txt");

doc.Delete();

doc.Print("doc\_deleted\_output.txt");

Document doc1 = new Document("mama моя лучшая");

Document doc2 = new Document("mama моя лучшая");

Document doc3 = new Document("mama моя самая лучшая");

Book book1 = new Book("Алиса в пограничье. Ты красотка");

Book book2 = new Book("Алиса в зазеркалье");

List<Document> archive = new List<Document>();

archive.Add(doc1);

archive.Add(doc2);

archive.Add(doc3);

archive.Add(book1);

archive.Add(book2);

foreach (Document item in archive)

{

((IEdit)item).Delete();

item.Print($"{DateTime.Now:yyyyMMdd}\_output.txt");

item.Delete();

item.Print($"{DateTime.Now:yyyyMMdd}\_deleted\_output.txt");

if (item is Book book)

{

book.ToBeContinue();

book.Print($"{DateTime.Now:yyyyMMdd}\_continued\_output.txt");

}

}

}

}

}

-------------------------------------------------------------------------------------------------------

using System.Runtime.Serialization.Json;

using System.Runtime.Serialization;

namespace \_4\_5

{

class MuchMoney : Exception

{

public MuchMoney(string message) : base(message)

{

}

}

class NoToDeleteFromWallet : Exception

{

public NoToDeleteFromWallet (string message) : base(message)

{

}

}

interface INumber

{

int Number { get; set; }

}

[DataContract]

class Bill : INumber

{

[DataMember]

private int number;

public int Number

{

get { return number; }

set

{

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

{

number = value;

}

else

{

throw new Exception("Неверное значение");

}

}

}

public Bill(int number)

{

Number = number;

}

}

[DataContract]

class Wallet<T> where T : INumber

{

[DataMember]

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

public void AddBill(Bill bill)

{

if (bills.Sum(x => x.Number) + bill.Number > 100)

{

throw new MuchMoney("Сумма купюр больше 100");

}

else

{

bills.Add(bill);

}

}

public void RemoveBill()

{

if (bills.Count == 0)

{

throw new NoToDeleteFromWallet("Купюр нет");

}

else

{

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

}

}

}

class Program

{

static void Main(string[] args)

{

Bill bill1 = new Bill(10);

Bill bill2 = new Bill(10);

Bill bill3 = new Bill(10);

Bill bill4 = new Bill(10);

Bill bill5 = new Bill(50);

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

wallet.AddBill(bill1);

wallet.AddBill(bill2);

wallet.AddBill(bill3);

wallet.AddBill(bill4);

wallet.AddBill(bill5);

wallet.RemoveBill();

var query = wallet.bills

.GroupBy(bill => bill.Number)

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

foreach (var item in query)

{

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

}

DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(Wallet<Bill>));

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

{

jsonFormatter.WriteObject(sw, wallet);

}

}

}

}

namespace examen

{

public class Card

{

public int number { get; }

public int month;

public int year { get; }

public int balance

{ get; set; }

public Card(int number, int month, int year, int balance)

{

this.balance = balance;

this.number = number;

this.month = month;

this.year = year;

}

public static Card operator +(Card card, int x)

{

card.balance += x;

return card;

}

public static Card operator -(Card card, int x)

{

card.balance -= x;

return card;

}

}

public class Computer : IComparable<Computer>

{

public string processor;

public string ram;

public int price;

public Computer(string proc, string r, int price)

{

processor = proc;

ram = r;

this.price = price;

}

public int CompareTo(Computer other)

{

if (this.price < other.price) return -1;

if (this.price > other.price) { return 1; }

return 0;

}

}

public static class StaticOperation

{

public static int WordCount(this string str)

{

int x = 0;

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

{

if (i == 0)

{

if (!char.IsLower(str[i]))

x++;

}

else if (!char.IsLower(str[i]) && str[i - 1] == ' ')

x++;

}

return x;

}

}

class Program

{

static void Main(string[] args)

{

Card card = new Card(11111, 11, 2020, 5000);

card = card + 15;

card = card - 10;

Console.WriteLine(card.balance);

Computer comp1 = new Computer("Intel", "8-gb", 200);

Computer comp2 = new Computer("Intel", "16-gb", 200);

Console.WriteLine(comp1.CompareTo(comp2));

string str = "Hekw Hkajd Ljad0";

Console.WriteLine(str.WordCount());

List<Computer> list = new List<Computer> { new Computer("Amd", "8-gb", 200), new Computer("Amd", "16-gb", 300), new Computer("Intel", "8-gb", 220), new Computer("Intel", "8-gb", 100), new Computer("Intel", "16-gb", 170), new Computer("Intel", "8-gb", 270) };

var res1 = list.Where(i => i.price < 200);

var res2 = list.Where(i => i.processor == "Amd");

var res3 = list.OrderBy(i => i.price);

string str1 = "";

foreach (var i in res1)

{

Console.WriteLine(i.processor + " " + i.price + " " + i.ram);

str1 += i.processor + " " + i.price + " " + i.ram + "\n";

}

Console.WriteLine();

str1 += "\n";

foreach (var i in res2)

{

Console.WriteLine(i.processor + " " + i.price + " " + i.ram);

str1 += i.processor + " " + i.price + " " + i.ram + "\n";

}

Console.WriteLine();

str1 += "\n";

foreach (var i in res3)

{

Console.WriteLine(i.processor + " " + i.price + " " + i.ram);

str1 += i.processor + " " + i.price + " " + i.ram + "\n";

}

Console.WriteLine();

str1 += "\n";

File.WriteAllText("A.txt", str1);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static Project1.Train;

namespace Project1

{

public abstract class Transsred

{

public abstract string Name { get; set; }

public abstract double Speed { get; set; }

public string Benz;

public int Year;

public abstract void Vichle();

}

public class Time

{

public int Hours { get; set; }

public int Minutes { get; set; }

public int Seconds { get; set; }

public Time() { }

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

{

if (hours < 0 || hours > 23 || minutes < 0 || minutes > 59 || seconds < 0 || seconds > 59)

{

throw new ArgumentOutOfRangeException("Error");

}

Hours = hours;

Minutes = minutes;

Seconds = seconds;

}

public static bool operator >(Time t1, Time t2)

{

if (t1.Hours > t2.Hours)

{

return true;

}

else if (t1.Hours == t2.Hours)

{

if (t1.Minutes > t2.Minutes)

{

return true;

}

else if (t1.Minutes == t2.Minutes)

{

return t1.Seconds > t2.Seconds;

}

}

return false;

}

public static bool operator <(Time t1, Time t2)

{

return !(t1 > t2) && !(t1.Equals(t2));

}

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

{

return false;

}

Time other = (Time)obj;

return Hours == other.Hours && Minutes == other.Minutes && Seconds == other.Seconds;

}

public override string ToString()

{

return base.ToString();

}

public void Print()

{

Console.WriteLine($"{Hours:D2}:{Minutes:D2}:{Seconds:D2}");

}

}

public class Train : Transsred

{

public double Number;

public int Vagon;

public float Passanger;

public Train(double number, int vagon, float passanger)

{

Number = number;

Vagon = vagon;

Passanger = passanger;

}

public override string Name { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public override double Speed { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public override void Vichle()

{

Console.WriteLine("train is go");

}

public class Car : Transsred

{

public string Brand { get; set; }

public string Model { get; set; }

public override double Speed { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public override string Name { get => throw new NotImplementedException(); set => throw new NotImplementedException(); }

public override void Vichle()

{

Console.WriteLine("Car is go");

}

public Car(string brand, string model, int year)

{

Brand = brand;

Model = model;

Year = year;

}

}

}

class Program

{

public static void Main(string[] args)

{

Time time = new Time(16, 18, 03);

time.Print();

Car car1 = new Car("Toyota", "Corolla", 2019);

car1.Vichle();

Train train = new Train(2003\_9, 6, 200);

train.Vichle();

List<Car> cars = new List<Car>

{

new Car("Toyota", "Coroll", 2002),

new Car("Ford", "Mustung", 2021),

new Car("BMW", "X5", 2004),

new Car("Audi", "A4", 2005)

};

foreach (var car in cars)

{

Console.WriteLine($"Brand: {car.Brand}, Model: {car.Model}, Year: {car.Year}");

}

List<Car> sortedCars = cars.OrderBy(car => car.Brand).ToList();

Console.WriteLine("\nSorts:");

foreach (var car in sortedCars)

{

Console.WriteLine($"Brand: {car.Brand}, Model: {car.Model}");

}

}

}

}

-------------------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Linq;

public class PDate

{

private int day;

private int month;

public int Day

{

get { return this.day; }

set

{

if (IsValidDay(value))

{

this.day = value;

}

else

{

Console.WriteLine("Ошибка при вводе дня");

}

}

}

public int Month

{

get { return this.month; }

set

{

if (IsValidMonth(value))

{

this.month = value;

}

else

{

Console.WriteLine("Ошибка при вводе месяца");

}

}

}

public int Year { get; set; }

public PDate(int day, int month)

{

this.day = day;

this.month = month;

}

private PDate() { }

public virtual void NextDay()

{

day++;

if (day > 30)

{

day = 1;

month++;

if (month > 12)

{

month = 1;

}

}

}

private bool IsValidDay(int day)

{

if (day >= 1 && day <= 31)

{

return true;

}

else

{

return false;

}

}

private bool IsValidMonth(int month)

{

if (month >= 1 && month <= 12)

{

return true;

}

else

{

return false;

}

}

}

public class PMDate : PDate

{

public int Year

{

get { return base.Year; }

set { base.Year = value; }

}

public PMDate(int day, int month, int year) : base(day, month)

{

Year = year;

}

public override void NextDay()

{

Day++;

if (Day == 1)

{

Month++;

if (Month == 1)

{

Year++;

}

}

}

public static bool operator >(PMDate date1, PMDate date2)

{

if (date1.Year > date2.Year)

{

return true;

}

else if (date1.Year == date2.Year)

{

if (date1.Month > date2.Month)

{

return true;

}

else if (date1.Month == date2.Month)

{

if (date1.Day > date2.Day)

{

return true;

}

}

}

return false;

}

public static bool operator<(PMDate date1, PMDate date2)

{

return !(date1 > date2);

}

}

public class Circle

{

private double \_radius;

public double Radius

{

get => \_radius;

set

{

if (value <= 0)

{

throw new ArgumentOutOfRangeException(nameof(value), "Radius must be positive.");

}

\_radius = value;

}

}

public double Area => Math.PI \* Radius \* Radius;

public Circle(double radius)

{

Radius = radius;

}

public override string ToString()

{

return $"Circle(Radius={Radius}, Area={Area})";

}

public override bool Equals(object obj)

{

if (obj is null)

{

return false;

}

if (!(obj is Circle circle))

{

return false;

}

return Radius.Equals(circle.Radius);

}

public override int GetHashCode()

{

return Radius.GetHashCode();

}

public int CompareTo(Circle other)

{

return Area.CompareTo(other.Area);

}

}

static class Program

{

static void Main(string[] args)

{

var c1 = new Circle(10);

var c2 = new Circle(5);

var c3 = new Circle(15);

var c4 = new Circle(7);

var c5 = new Circle(5);

Console.WriteLine(c1);

Console.WriteLine(c2);

//Console.WriteLine(c1.Equals(c5));

Console.WriteLine(c1.GetHashCode() == c2.GetHashCode());

Console.WriteLine(c1.CompareTo(c2));

Console.WriteLine(c1.Area);

Console.WriteLine(c2.Area);

List<Circle> circles = new List<Circle>() { c1, c2, c3, c4 };

circles.Sort((a, b) => a.Area.CompareTo(b.Area));

foreach (var circle in circles)

{

Console.WriteLine(circle);

}

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

PDate date1 = new PDate(15, 6);

PDate date3 = new PDate(14, 7);

Console.WriteLine($"Date1: {date1.Day}/{date1.Month}");

date1.NextDay();

Console.WriteLine($"Next day: {date1.Day}/{date1.Month}");

PMDate date2 = new PMDate(28, 2, 2024);

Console.WriteLine($"Date2: {date2.Day}/{date2.Month}/{date2.Year}");

date2.NextDay();

date2.NextDay();

date2.NextDay();

Console.WriteLine($"Next day: {date2.Day}/{date2.Month}/{date2.Year}");

}

}

using System.Xml;

using Newtonsoft.Json;

namespace Ekzamen

{

public class Circle : IComparable

{

public int Radius { get; set; }

public string Color { get; set; }

public Circle(int radius, string color)

{

this.Radius = radius;

this.Color = color;

}

public double Area(int radius)

{

double area = Math.Pow(radius, 2) \* Math.PI;

return area;

}

public override string ToString()

{

return $"Круг с радиусом {Radius} и цветом {Color}";

}

public bool Equals(Circle circle)

{

if (circle == null)

{

return false;

}

else

{

return true;

}

}

public int CompareTo(object obj)

{

if (obj == null || GetType() != obj.GetType())

{

return 1;

}

Circle other = (Circle)obj;

return Radius.CompareTo(other.Radius);

}

public int CompareTo(Circle circle)

{

return Radius.CompareTo(circle.Radius);

}

}

public class NTDate : NDate

{

public int Year

{

get { return base.Year; }

set { base.Year = value; }

}

public NTDate(int day, int month, int year) : base(day, month)

{

Year = year;

}

public override void NextDay()

{

Day++;

if (Day == 1)

{

Month++;

if (Month == 1)

{

Year++;

}

}

}

public static bool operator >(NTDate date1, NTDate date2)

{

if (date1.Year > date2.Year)

{

return true;

}

else if (date1.Year == date2.Year)

{

if (date1.Month > date2.Month)

{

return true;

}

else if (date1.Month == date2.Month)

{

if (date1.Day > date2.Day)

{

return true;

}

}

}

return false;

}

public static bool operator <(NTDate date1, NTDate date2)

{

return !(date1 > date2);

}

}

public class NDate

{

private int day;

private int month;

public int Day

{

get { return this.day; }

set

{

if (IsValidDay(value))

{

this.day = value;

}

else

{

Console.WriteLine("Не правильный день (1-31)");

}

}

}

private bool IsValidDay(int day)

{

if (day >= 1 && day <= 31)

{

return true;

}

else

{

return false;

}

}

public int Month

{

get { return this.month; }

set

{

if (IsValidMonth(value))

{

this.month = value;

}

else

{

Console.WriteLine("Не правильный месяц (1-12)");

}

}

}

public int Year { get; set; }

public NDate(int day, int month)

{

this.day = day;

this.month = month;

}

private NDate()

{

}

public virtual void NextDay()

{

day++;

if (day > 31)

{

day = 1;

month++;

if (month > 12)

{

month = 1;

}

}

}

private bool IsValidMonth(int month)

{

if (month >= 1 && month <= 12)

{

return true;

}

else

{

return false;

}

}

}

public static class Program

{

static void Main()

{

List<Circle> circles = new List<Circle>()

{

new Circle(2, "red"),

new Circle(3, "green"),

new Circle(3, "yellow"),

new Circle(4, "purple"),

new Circle(5, "green"),

new Circle(6, "red")

};

var sortedCircles = circles

.Skip(3)

.OrderBy(c => c.Color)

.GroupBy(c => c.Color);

foreach (var group in sortedCircles)

{

Console.WriteLine("Цвет: " + group.Key);

foreach (var circle in group)

{

Console.WriteLine("Радиус: " + circle.Radius);

}

Console.WriteLine();

}

/\*XmlSerializer serializer = new XmlSerializer(typeof(List<Circle>));

using (StreamWriter writer = new StreamWriter("circles.xml"))

{

serializer.Serialize(writer, sortedCircles);

}\*/

string json = JsonConvert.SerializeObject(sortedCircles);

Console.WriteLine(json);

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

NDate date = new NDate(15, 10);

Console.WriteLine("День: " + date.Day);

Console.WriteLine("Месяц: " + date.Month);

date.NextDay();

Console.WriteLine("Следующий день: " + date.Day + "." + date.Month);

NTDate date1 = new NTDate(15, 10, 2023);

NTDate date2 = new NTDate(20, 10, 2023);

Console.WriteLine("Дата 1: " + date1.Day + "." + date1.Month + "." + date1.Year);

Console.WriteLine("Дата 2: " + date2.Day + "." + date2.Month + "." + date2.Year);

if (date1 > date2)

{

Console.WriteLine("Дата 1 позже Дата 2");

}

else if (date1 < date2)

{

Console.WriteLine("Дата 1 раньше Дата 2");

}

else

{

Console.WriteLine("Дата 1 и Дата 2 одинаковы");

}

}

}

}

using System;

using System.Linq;

public interface IPay

{

void Pay(int amount, ExDate exDate);

}

public class ExDate

{

public int Month { get; set; }

public int Year { get; set; }

public ExDate(int month, int year)

{

Month = month;

Year = year;

}

}

public class Card : IPay

{

private int balance;

private ExDate expirationDate;

private string number;

public Card(int balance, ExDate expirationDate, string number)

{

this.balance = balance;

this.expirationDate = expirationDate;

this.number = number;

}

public void Pay(int amount, ExDate exDate)

{

if (balance - amount < -100)

{

throw new InvalidOperationException("Insufficient funds.");

}

Console.WriteLine($"Payment of {amount} made successfully with Card {number}");

balance -= amount;

}

public int GetBalance()

{

return balance;

}

public static explicit operator IPay(Card card)

{

return new CardProxy(card);

}

}

public class CardProxy : IPay

{

private Card card;

public CardProxy(Card card)

{

this.card = card;

}

public void Pay(int amount, ExDate exDate)

{

if (card.GetBalance() - amount < -100)

{

throw new InvalidOperationException("Insufficient funds.");

}

card.Pay(amount, exDate);

}

}

class Program

{

static void Main()

{

Card[] cards = new Card[]

{

new Card(150, new ExDate(12, 23), "1111-2222-3333-4444"),

new Card(200, new ExDate(10, 24), "2222-3333-4444-5555"),

new Card(50, new ExDate(8, 22), "3333-4444-5555-6666")

};

var cardWithMaxBalance = cards.OrderByDescending(c => c.GetBalance()).First();

Console.WriteLine($"Card with maximum balance: {cardWithMaxBalance.GetBalance()}, Card Number: {cardWithMaxBalance}");

}

}

using System;

using System.Collections.Generic;

interface IAction<T>

{

void Add(T item);

void Remove(T item);

void Print();

void Clear();

}

class Vector<T> : IAction<T>

{

private List<T> collection;

public Vector()

{

collection = new List<T>();

}

public void Add(T item)

{

collection.Add(item);

}

public void Remove(T item)

{

if (collection.Contains(item))

{

collection.Remove(item);

}

else

{

Console.WriteLine($"Item {item} not found.");

}

}

public void Print()

{

foreach (var item in collection)

{

Console.WriteLine(item);

}

}

public void Clear()

{

collection.Clear();

}

}

class Student

{

public string Name { get; set; }

public int Age { get; set; }

public Student(string name, int age)

{

Name = name;

Age = age;

}

public override string ToString()

{

return $"Name: {Name}, Age: {Age}";

}

}

class Program

{

static void Main()

{

try

{

// Целочисленный тип

Vector<int> intVector = new Vector<int>();

intVector.Add(1);

intVector.Add(2);

intVector.Add(3);

Console.WriteLine("Integer Vector:");

intVector.Print();

// Студент

Vector<Student> studentVector = new Vector<Student>();

studentVector.Add(new Student("John", 20));

studentVector.Add(new Student("Alice", 22));

Console.WriteLine("\nStudent Vector:");

studentVector.Print();

}

catch (Exception ex)

{

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

}

finally

{

Console.WriteLine("Finally block executed.");

}

}

}

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());

}

}

}

using System.Text.Json;

using System.Text.Json.Serialization;

namespace \_7\_8\_3

{

internal class Program

{

[Serializable]

class Time : IComparable

{

private int hours = 0, minutes = 0, seconds = 0;

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

{

CheckHours = hours;

CheckMinutes = minutes;

CheckSeconds = seconds;

}

public int CheckHours

{

get { return hours; }

set

{

if (hours < 0 || hours >= 24) Console.WriteLine("Значение часов некорректно.");

else hours = value;

}

}

public int CheckMinutes

{

get { return minutes; }

set

{

if (minutes < 0 && minutes >= 60) Console.WriteLine("Значение минут некорректно.");

else minutes = value;

}

}

public int CheckSeconds

{

get { return seconds; }

set

{

if (seconds < 0 && seconds >= 60) Console.WriteLine("Значение секунд некорректно.");

else seconds = value;

}

}

public int CompareTo(object? obj)

{

if (obj is Time x)

{

if ((x.CheckHours == this.CheckHours) && (x.CheckMinutes == this.CheckMinutes))

{

return 0;

}

if (this.CheckHours > x.CheckHours)

{

return 1;

}

else

{

return -1;

}

}

else

{

throw new Exception("Неверно!");

}

}

}

static void Main(string[] args)

{

Time time1 = new Time(3, 1, 20);

Time time2 = new Time(3, 14, 24);

Time[] arr = new Time[]

{

new Time(20, 19, 18),

new Time(2, 13, 28),

new Time(13, 28, 1),

new Time(15, 59, 29),

new Time(22, 1, 9),

new Time(8, 45, 32)

};

var night = arr.Where(x => (x.CheckHours>=0 && x.CheckHours <= 5)).OrderBy(x => x.CheckHours).ToArray();

var morning = arr.Where(x => (x.CheckHours > 5 && x.CheckHours <= 12)).OrderBy(x => x.CheckHours).ToArray();

var afternoon = arr.Where(x => (x.CheckHours > 12 && x.CheckHours <= 18)).OrderBy(x => x.CheckHours).ToArray();

var evening = arr.Where(x => (x.CheckHours > 18 && x.CheckHours < 24)).OrderBy(x => x.CheckHours).ToArray();

Console.WriteLine("night");

foreach (var x in night) { Console.WriteLine(x.CheckHours); }

Console.WriteLine("morning");

foreach (var x in morning) { Console.WriteLine(x.CheckHours); }

Console.WriteLine("afternoon");

foreach (var x in afternoon) { Console.WriteLine(x.CheckHours); }

Console.WriteLine("evening");

foreach (var x in evening) { Console.WriteLine(x.CheckHours); }

using (StreamWriter sw = new StreamWriter("time.txt"))

{

sw.WriteLine("night");

foreach (var x in night) { sw.WriteLine(x.CheckHours); }

sw.WriteLine("morning");

foreach (var x in morning) { sw.WriteLine(x.CheckHours); }

sw.WriteLine("afternoon");

foreach (var x in afternoon) { sw.WriteLine(x.CheckHours); }

sw.WriteLine("evening");

foreach (var x in evening) { sw.WriteLine(x.CheckHours); }

}

Time test = new Time(2, 3, 4);

string json = JsonSerializer.Serialize(test);

Console.WriteLine(json);

Time? test2 = JsonSerializer.Deserialize<Time>(json);

try

{

Console.WriteLine(time1.CompareTo(time2));

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

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);

}

}

}