**ЛАБА 2**

using System;

using System.Linq;

using System.Text;

Console.WriteLine($"bool: {boolValue}\nbyte: {byteValue}\nsbyte: {sbyteValue}\nchar: {charValue}");

//c. упаковка и распаковка

int x = 255;

Object obj = x;

x = (int)obj;

long y = 5786;

Object obje = y;

y = (long)obje;

short z = 877;

Object objec = z;

z = (short)objec;

//e. nullable

double? pi = 3.14;

int? intnull = null;

// СТРОКИ

string str1 = "Hello";

string str2 = "World";

Console.WriteLine($"литералы { ((str1 == str2) ? "" : "не")} равны\n");

string string1 = new string('e', 8);

string string2 = "hello world";

string string3 = new string('!', 3);

//сцепление строк

string string4 = string.Concat(string2, string3);

Console.WriteLine("конкатенация строк 2 и 3: " + string4);

//копирование строк

string string5 = string.Copy(string1);

Console.WriteLine("копирование строки 1: " + string5);

//подстрока с 2 позиции длинной 3 символа

string string6 = string2.Substring(2, 3);

Console.WriteLine("строка 4 (вырезано 2 символа из строки 2): " + string6);

//вставка стр 6 в стр 1 с позиции 2

string string7 = string1.Insert(2, string6);

Console.WriteLine("вставка вырезанного в строку 1 с позиции 2: " + string7);

string string8 = string2.Replace(string6, "aaaaaaaaa");

Console.WriteLine("замена в строке 2 стрки 4 на аааааа: " + string8);

//разбиение строки на массив слов

string[] words = string2.Split(new Char[] { ' ', ',', '.', ':', '\t' });

Console.WriteLine("строка 2 разбитая на слова:");

Array.ForEach(words, Console.WriteLine);

string nullString = null;

string emptyString = string.Empty;

Console.WriteLine($"Длина строки 2 равна {emptyString.Length}");

var stringBuilder = new StringBuilder("woah stringbuilder");

Console.WriteLine($"Первоначальная строка: {stringBuilder}");

Console.WriteLine($"Строка после удаления: {stringBuilder.Remove(3, 3)}");

stringBuilder.Append("!");

stringBuilder.Insert(0, "?");

Console.WriteLine($"Строка после вставки символов: {stringBuilder}");

Console.WriteLine("\ta. двумерный массив в виде матрицы: ");

int[,] arr = new int[3, 3] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

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

{

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

{

Console.Write(arr[ic, j] + " ");

}

Console.WriteLine();

}

//массив строк

string[] arrHPbooks = { "Philosopher's Stone", "Chamber of Secrets", "Prisoner of Azkaban", "Goblet of Fire", "Order of the Phoenix", "Half-Blood Prince", "Deathly Hallows" };

Array.ForEach(arrHPbooks, Console.WriteLine);

Console.WriteLine("\nДлина массива: " + arrHPbooks.Length);

Console.Write("\nВведите позицию для вставки строки: ");

var index = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Введите строку: ");

var strToReplace = Console.ReadLine();

arrHPbooks[index] = strToReplace;

Console.WriteLine("\nНовый массив: ");

Array.ForEach(arrHPbooks, Console.WriteLine);

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

// КОРТЕЖИ

(int, string, char, string, ulong) tuple = (63, "Woof", 'A', "Гав", 127);

Console.WriteLine($"кортеж: {tuple}");

Console.WriteLine($"1, 3, 4 элементы кортежа: {tuple.Item1}, {tuple.Item3} и {tuple.Item4}.");

//d. Сравните два кортежа.

Console.WriteLine($"(4, 5) == (4, 5): {(4, 5) == (4, 5)}");

// локальная функция, принимает массив интов и строку, возвращает кортеж

(int, int, int, char) Task5(int[] arr, string str)

{

return (arr.Max(), arr.Min(), arr.Sum(), str.First());

}

Array.ForEach(nums, Console.WriteLine);

checked

{

Console.WriteLine("checked:");

int check = int.MaxValue;

Console.WriteLine($"int.MaxValue checked: {check}");

//Console.WriteLine($"int.MaxValue + 1 checked: {check+1}"); //исключение: переполнение

}

unchecked

{

Console.WriteLine("unchecked:");

int uncheck = int.MaxValue;

Console.WriteLine($"int.MaxValue unchecked: {uncheck}");

Console.WriteLine($"int.MaxValue + 1 unchecked: {uncheck+10}");

}

**ЛАБА 3**

PHONE.CS:

using System;

using System.Collections.Generic;

using System.Text;

namespace ootp\_lab3

{

partial class Phone

{

public readonly int id;

public string Surname { get; set; }

public string FirstName { get; set; }

public string SecondName { get; set; }

public string Address { get; set; }

public long Card;

public long CardNumber // 0000 0000 0000 0000

{

get => Card;

set

{

if (value < 999999999999999 && value > 9999999999999999)

{

Console.WriteLine("Неправильно введен номер платежной карты");

}

else

{

Card = value;

}

}

}

public float Debit { get; set; }

public float Credit { get; set; }

public int IntercityCalls { get; set; }

public int CityCalls { get; set; }

public const byte aaaaaa = 1;

static int PhonesAmount = 0; // счетчик количества абонентов

//конструктор с параметрами по умолчанию

public Phone(string surname = "Unknown", string name = "Unknown", string second = "Unknown", string address = "Unknown", long card = 0, float debit = 0, float credit = 0, int city = 0, int intercity = 0)

{

FirstName = name;

Surname = surname;

SecondName = second;

Address = address;

CardNumber = card;

Debit = debit;

Credit = credit;

IntercityCalls = intercity;

CityCalls = city;

PhonesAmount++;

id = Math.Abs(this.GetHashCode());

}

//конструктор с параметрами

public Phone(Phone a) : this(a.Surname, a.FirstName, a.SecondName, a.Address, a.CardNumber, a.Debit, a.Credit, a.IntercityCalls, a.CityCalls)

{

id = Math.Abs(this.GetHashCode());

}

static Phone()

{

Console.WriteLine("Добавлен абонент");

PhonesAmount++;

}

private Phone()

{

Console.WriteLine("Будущий абонент");

PhonesAmount++;

}

public float CheckBalance()

{

return this.Debit - this.Credit;

}

public void PrintInfo()

{

Console.WriteLine();

Console.WriteLine("\tID АБОНЕНТА: " + id);

Console.WriteLine("ФИО: " + Surname + " " + FirstName + " " + SecondName);

Console.WriteLine("Адрес: " + Address + "\nНомер карты: " + CardNumber + "\nБаланс: " + this.CheckBalance());

Console.WriteLine("Время городских звонков: " + CityCalls + " минут \nВремя междугородных звонков: " + IntercityCalls + " минут");

}

}

}

PARTIALPHONE.CS

using System;

using System.Collections.Generic;

using System.Text;

namespace ootp\_lab3

{

partial class Phone

{

// переопределение функций

public override int GetHashCode() => HashCode.Combine(Surname, FirstName, SecondName, Address, CardNumber);

public override bool Equals(object obj)

{

if(obj is Phone ph) return (this.FirstName == ph.FirstName && this.Surname == ph.Surname && this.SecondName == ph.SecondName &&

this.Address == ph.Address && this.CardNumber == ph.CardNumber && this.CheckBalance() == ph.CheckBalance() && this.CityCalls == ph.CityCalls && this.IntercityCalls == ph.IntercityCalls);

return false;

}

public override string ToString() => $"Абонент {id}: {Surname} {FirstName} {SecondName}";

}

}

PROGRAM.CS

using System;

namespace ootp\_lab3

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

Phone ph1 = new Phone("Ермаков", "Виссарион", "Еремеевич", "г. Ловозеро, ул. Черносельская, дом 38, квартира 456", 5256261555541992, 110, 20, 40, 4);

Phone ph2 = new Phone("Васильев", "Гарри", "Дамирович", "г. Сольцы, ул. Лермонтовский пр-кт, дом 110, квартира 370", 5467592824767538, 230, 65, 120, 74);

Phone ph3 = new Phone("Потапов", "Тихон", "Львович", "г. Троицкое, ул. Украинский б-р, дом 167, квартира 349", 5191100508125388, 50, 5, 39);

Phone ph4 = new Phone("Русаков", "Емельян", "Сергеевич", "г. Новосергиевка, ул. Гапсальская, дом 85, квартира 234", 5445188636065679, 45, 50, 3);

Phone ph5 = new Phone("Федосеев", "Эдуард", "Яковлевич", "г. Угловское, ул. Тверской-Ямской 1-й пер, дом 172, квартира 209", 4154229896887545, 163, 39, 187, 32);

var Subscribers = new[] { ph1, ph2, ph3, ph4, ph5 };

Console.WriteLine("Список абонентов:");

foreach(Phone ph in Subscribers)

{

ph.PrintInfo();

}

Console.WriteLine("\nАбоненты, пользовавшиеся междугородной связью:\n");

foreach (Phone ph in Subscribers)

{

if (ph.IntercityCalls > 0)

Console.WriteLine(ph.ToString());

}

Console.Write("\nВывести сведения об абонентах, у которых время внутригородских разговоров превышает: ");

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

Console.WriteLine($"\nАбоненты, у которых время внутригородских разговоров превышает {check} минут:\n");

foreach (Phone ph in Subscribers)

{

if (ph.CityCalls > check)

Console.WriteLine(ph.ToString());

}

Console.WriteLine($"\nСравнение абонентов 1 (ID:{ph1.id}) и 2 (ID:{ph2.id}):");

if (ph1.Equals(ph2)) Console.WriteLine("Абоненты равны");

else Console.WriteLine("Абоненты не равны");

}

}

}

**ЛАБА 4**

OWNER.CS

using System;

using System.Diagnostics;

namespace ootp\_lab04

{

public partial class Set

{

public static Owner OwnerInfo { get; private set; }

public static Date CreationDate { get; private set; }

static Set()

{

OwnerInfo = new Owner("61201002", "Natalie", "BSTU");

CreationDate = "26.11.2020";

}

public class Date

{

private ushort day, month;

public ushort Day

{

get => day;

set

{

if (value > 31)

{

day = 31;

}

else

{

day = value;

}

}

}

public ushort Month

{

get => month;

set

{

if (value > 12)

{

month = 12;

}

else

{

month = value;

}

}

}

public int Year { get; set; }

public Date(string date)

{

string[] tokens = date.Split('.', StringSplitOptions.RemoveEmptyEntries);

if (tokens.Length != 3)

{

Debug.WriteLine("парсинг даты сломался");

}

else

{

Day = ushort.Parse(tokens[0]);

Month = ushort.Parse(tokens[1]);

Year = int.Parse(tokens[2]);

}

}

public Date(ushort day, ushort month, int year)

=> (Day, Month, Year) = (day, month, year);

public static implicit operator Date(string s) => new Date(s);

public override string ToString() => $"{Day}.{Month}.{Year}";

public override bool Equals(object o) =>

(o is Date date) && (date.Day, date.Month, date.Year) ==

(Day, Month, Year);

public override int GetHashCode() => HashCode.Combine(Day, Month, Year);

}

public class Owner

{

public string GithubID { get; private set; }

public string Name { get; private set; }

public string Organization { get; private set; }

public Owner(string id, string name, string organisation)

=> (GithubID, Name, Organization) = (id, name, organisation);

public override string ToString() => $"ID: {GithubID}, Имя: {Name}, Организация: {Organization}";

}

}

}

PROGRAM.CS

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

/\* Класс - множество Set. Дополнительно перегрузить следующие операции:

> - проверка на принадлежность (типа операции in множественного типа),

\* - пересечение множеств;

< - проверка на подмножество;

явный Date() - приведение типа

Методы расширения:

1) Выделение первого числа, содержащегося в строке

2) Удаление положительных элементов из множества \*/

namespace ootp\_lab04

{

public static class StatisticalOperations

{

public static int GetFirstNumber(this string s)

{

var tokens = s.Split();

foreach (var token in tokens)

{

if (int.TryParse(token, out int res))

{

return res;

}

}

return default;

}

public static Set RemovePositiveElements(this Set s)

=> new Set(s.Where(el => el < 0));

public static int SumOfElements(this Set s) => s.Sum();

public static int MinMaxDifference(this Set s)

{

int max = s[0];

int min = s[0];

foreach (int el in s)

{

if (el > max)

{

max = el;

}

if (el < min)

{

min = el;

}

}

return max - min;

}

public static int ElementCount(this Set s) => s.Count();

}

public partial class Set : IEnumerable<int>

{

List<int> container;

public Set(params int[] values) : this(values.AsEnumerable())

{

}

public Set(IEnumerable<int> values)

=> container = values.Distinct().ToList();

public int this[int index]

{

get => container[index];

set => container[index] = value;

}

public void Add(int el)

{

if (!container.Contains(el))

{

container.Add(el);

}

}

public void Add(params int[] els) => container = container.Union(els).ToList();

public static Set operator +(Set l, Set r) => new Set(l.Concat(r));

public static Set operator +(Set s, int el) => new Set(s.Append(el));

public static Set operator +(int el, Set s) => new Set(s.Prepend(el));

public static Set operator \*(Set l, Set r) => new Set(l.Intersect(r));

public static bool operator >(int el, Set s) => s.Contains(el);

public static bool operator <(int el, Set s) => !(el > s);

public static bool operator >(Set subset, Set set)

{

bool isSubset = true;

foreach (var element in subset)

{

if (!set.Contains(element))

{

isSubset = false;

break;

}

}

return isSubset;

}

public static bool operator <(Set set, Set subset)

{

bool isSubset = true;

foreach (var element in subset)

{

if (!set.Contains(element))

{

isSubset = false;

break;

}

}

return isSubset;

}

public static explicit operator Date(Set set) => new Date((ushort)set[0], (ushort)set[1], set[2]);

public IEnumerator<int> GetEnumerator() => ((IEnumerable<int>)container).GetEnumerator();

IEnumerator IEnumerable.GetEnumerator() => container.GetEnumerator();

public override string ToString() => '{' + string.Join(", ", this) + '}';

}

class Program

{

static void Main()

{

var set1 = new Set { 1, 2, 5, 2, 13, 2, 4 };

var set2 = new Set { 7, 6, 5, 2, 4, 1 };

Console.WriteLine($"{set1} + {set2} = {set1 + set2}");

Console.WriteLine($"{set1} \* {set2} = {set1 \* set2}");

Console.WriteLine($"{set1} + 12 = {set1 + 12}");

Console.WriteLine($"12 + {set1} = {12 + set1}");

Console.WriteLine($"Принадлежность: 2 > {set1} = {2 > set1}");

var subset = new Set { 1, 2, 3 };

var set = new Set { 1, 2, 3, 4, 5 };

Console.WriteLine($"Подмножество: {subset} > {set} = {subset > set}");

Console.WriteLine($"(Date){subset} = {(Set.Date)subset}");

var set3 = new Set { 1, -9, 6, 4, -3, 5, 2 };

Console.WriteLine($"{set3}.RemovePositiveElements() = {set3.RemovePositiveElements()}");

var numString = "abc 12 djvr 89 jefl 1 jf";

Console.WriteLine($"\"{numString}\".GetFirstNumber() = {numString.GetFirstNumber()}");

}

}

}

**ЛАБА 4\_2**

OWNER.CS

using System;

namespace Lab\_4

{

public class Owner

{

private readonly int id;

private readonly string fio;

public Owner(int id, string fio)

{

this.id = id;

this.fio = fio;

}

public void GetInfo()

{

Console.WriteLine($"Owner - ID: {id}, FIO: {fio}");

}

}

}

SET.CS

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lab\_4

{

public class MySet

{

private readonly Owner owner;

public HashSet<String> collection;

private readonly Date date;

public int Size;

public MySet(int ownerId, string ownerFIO)

{

this.owner = new Owner(ownerId, ownerFIO);

this.collection = new HashSet<string>();

this.date = new Date();

}

public HashSet<string> GetHash()

{

return collection;

}

public void Show()

{

foreach(string item in collection)

{

Console.WriteLine(item);

}

}

public Owner GetOwner()

{

return owner;

}

public int GetSize()

{

int size = 0;

foreach (string item in collection)

{

size++;

}

return size;

}

public void AddItem(string item)

{

collection.Add(item);

}

public class Date

{

public readonly DateTime time;

public Date()

{

time = DateTime.Now;

}

}

public string GetItemByIndex(int index)

{

if (index > this.GetSize() - 1)

throw new Exception("GetItemByIndex: OutOfRange");

int size = -1;

foreach (string item in collection)

{

size++;

if (size == index)

return item;

}

return "";

}

#region Overload

public static MySet operator - (MySet set, string item)

{

set.collection.Remove(item);

return set;

}

public static MySet operator \* (MySet set, MySet set2)

{

set.collection.IntersectWith(set2.collection);

return set;

}

public static bool operator <(MySet set, MySet set2)

{

return set.GetSize() < set2.GetSize();

}

public static bool operator >(MySet set, MySet set2)

{

if (set.collection.IsSubsetOf(set2.collection))

return true;

else

return false;

}

public static MySet operator &(MySet set, MySet set2)

{

set.collection.Except(set2.collection);

return set;

}

#endregion

}

}

STATISTICALOPERATIONS.CS

using System;

using System.Collections.Generic;

namespace Lab\_4

{

static class StatisticOperations

{

public static int Max(this MySet set)

{

int len = set.GetItemByIndex(0).Length;

foreach (string item in set.GetHash())

{

if (len < item.Length)

len = item.Length;

}

return len;

}

public static int Min(this MySet set)

{

int len = set.GetItemByIndex(0).Length;

foreach (string item in set.GetHash())

{

if (len > item.Length)

len = item.Length ;

}

return len;

}

public static int Dif(this MySet set)

{

return Max(set) - Min(set);

}

public static int CountOfWords(this string str)

{

int count = 0;

string[] words = str.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

count += words.Length;

return count;

}

public static int Sum(this MySet set)

{

int len = 0;

foreach(string item in set.GetHash())

{

len += item.Length;

}

return len;

}

public static void CommaAfterWord(this MySet set)

{

int len = set.GetSize();

HashSet<string> res = new HashSet<string>();

string buf = "";

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

{

buf = set.GetItemByIndex(i);

res.Add(buf);

res.Add(",");

}

set.collection = res;

}

public static void RemoveSpace(this MySet set)

{

set.collection.Remove("");

}

}

}

PROGRAM.CS

using System;

namespace Lab\_4

{

class Program

{

static void Main(string[] args)

{

MySet set = new MySet(1, "SYO");

set.AddItem("Max");

set.AddItem("Nick");

MySet set2 = new MySet(2, "BEI");

set2.AddItem("Max");

set2.AddItem(" ");

set2.AddItem("Lia");

set2.AddItem("Nick");

MySet set3 = new MySet(3, "LIO");

set3.AddItem("Jio");

set3.AddItem("Mari");

MySet set4 = new MySet(4, "KPF");

set4.AddItem("Jhon");

set4.AddItem("");

set4.AddItem("Leo");

Console.WriteLine("--------- Перегрузка оператора - ---------");

string name = "Jio";

set3 = set3 - name;

set3.Show();

Console.WriteLine("--------- Перегрузка оператора \* ------------");

set2 = set2 \* set;

set2.Show();

Console.WriteLine("--------- Перегрузка оператора & ----------------");

set3 = set3 & set;

set3.Show();

Console.WriteLine("--------- Перегрузка оператора > ----------------");

Console.WriteLine(set > set2);

Console.WriteLine("--------- Перегрузка оператора < ----------------");

Console.WriteLine(set < set2);

Console.WriteLine("------------- Расширение Dif ----------------");

int r = set2.Dif();

Console.WriteLine(r);

Console.WriteLine("---------------------------------------------");

string abc = "awdg fseol ajes";

Console.WriteLine($"Count of words: {abc.CountOfWords()}");

Console.WriteLine("---------------------------------------------");

int r2 = set3.Sum();

Console.WriteLine($"Sum lenght of all words: {r2}");

Console.WriteLine("---------------------------------------------");

set4.Show();

set4.RemoveSpace();

set4.Show();

Console.WriteLine("---------------------------------------------");

set4.CommaAfterWord();

set4.Show();

}

}

}

**ЛАБА 5**

CLASSES.CS

using System;

namespace Lab\_5

{

class Printer

{

public void iAmPrinting(Plant someobj)

{

Console.WriteLine("Тип объекта = " + someobj.ToString());

}

}

abstract class Plant

{

protected int numOfPlants = 0;

public override string ToString() => "Plant";

public Plant() { }

public virtual void Pour(int litters)

{

Console.WriteLine($"Вы полили растение {litters} литрами воды");

}

public abstract void ToPlant();

public void GetPlants() => Console.WriteLine($"Всего {numOfPlants} растеный вида {ToString()}");

}

sealed class Bush : Plant

{

public override string ToString() => "Bush";

public override void ToPlant() => Console.WriteLine("Вы посадили куст", numOfPlants += 1);

}

class Flower : Plant

{

public override string ToString() => "Flower";

public override void ToPlant() => Console.WriteLine("Вы посадили цветок", numOfPlants += 1);

public override bool Equals(object obj)

{

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

return false;

Flower other = (Flower)obj;

return (numOfPlants == other.numOfPlants);

}

public override int GetHashCode()

{

return DateTime.Now.Millisecond;

}

}

class Rose : Flower, IPaper

{

public override string ToString() => "Rose";

public override void ToPlant() => Console.WriteLine("Вы посадили розу", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали букет роз");

public void PackIn() => Console.WriteLine("Вы обернули букет в бумагу");

}

class Gladiolus : Flower, IPaper

{

public override string ToString() => "Gladiolus";

public override void ToPlant() => Console.WriteLine("Вы посадили гладиолус", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали букет из гладиолусов");

public void PackIn() => Console.WriteLine("Вы обернули букет в бумагу");

}

class Cactus : Flower, IPot

{

public override string ToString() => "Cactus";

public override void ToPlant() => Console.WriteLine("Вы посадили кактус (abstract)", numOfPlants += 1);

void IPot.ToPlant() => Console.WriteLine("Вы посадили кактус (interface)", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали кактус");

public void PutInPot() => Console.WriteLine("Вы поместили кактус в горшок");

}

}

INTERFACES.CS

namespace Lab\_5

{

public interface IPaper : IBouquet

{

void PackIn();

}

public interface IBouquet

{

void Collect();

}

public interface IPot : IBouquet

{

void PutInPot();

void ToPlant();

}

}

COLLECTION.CS

using System;

using System.Collections.Generic;

namespace Lab\_5

{

partial class CollectionType<T> : IGereric<T> where T:class

{

private int size;

public List<T> list;

public CollectionType()

{

list = new List<T>();

}

public void Add(T item)

{

list.Add(item);

size++;

}

public void Delete(int index)

{

try

{

list.RemoveAt(index);

size--;

}

catch

{

Console.WriteLine($"Выход за пределы, используйте числа в диапазоне (0-{size})");

}

}

public void Show()

{

foreach (T item in list)

{

Console.WriteLine(item);

}

}

public int GetSize()

{

return size;

}

}

}

INTERFACE.CS

namespace Lab\_5

{

public interface IGereric<T>

{

void Add(T item);

void Delete(int index);

void Show();

}

}

FILESTREAM.CS

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace Lab\_5

{

partial class Program

{

public static void ParseFile(CollectionType<Plant> objColl)

{

using (StreamReader stream = new StreamReader(@"K:\Uni\OOP-Labs-master\Lab\_5\Lab\_5\5\_lab.txt"))

{

while (stream.ReadLine() is string line)

{

switch (line)

{

case "Rose": objColl.Add(new Rose());

break;

case "Bush": objColl.Add(new Bush());

break;

case "Gladiolus": objColl.Add(new Gladiolus());

break;

case "Cactus": objColl.Add(new Cactus());

break;

}

}

}

}

}

partial class CollectionType<T>

{

public void WriteFile(CollectionType<Plant> objColl)

{

List<Plant> res = objColl.list;

using(StreamWriter write = new StreamWriter(@"K:\Uni\OOP-Labs-master\Lab\_5\Lab\_5\Write.txt"))

{

foreach(Plant item in res)

{

write.WriteLine(item);

}

}

}

}

}

PROGRAM.CS

using System;

namespace Lab\_5

{

partial class Program

{

static void Main(string[] args)

{

Console.WriteLine("-----------Розы-------------");

Rose rose = new Rose();

rose.ToPlant();

rose.GetPlants();

rose.ToPlant();

rose.ToPlant();

rose.GetPlants();

rose.Pour(3);

if (rose is Plant)

{

Console.WriteLine($"{rose} is plant");

}

Console.WriteLine("-----------Кактусы-------------");

Cactus cactus = new Cactus();

IPot cactus\_2 = cactus;

cactus\_2.ToPlant();

cactus.ToPlant();

cactus.GetPlants();

cactus.Collect();

cactus.PutInPot();

Console.WriteLine("-----------Гладиолусы-------------");

Gladiolus gladiolus = new Gladiolus();

gladiolus.ToPlant();

gladiolus.ToPlant();

gladiolus.Pour(2);

gladiolus.Collect();

gladiolus.PackIn();

Printer printer = new Printer();

printer.iAmPrinting(cactus);

CollectionType<string> obj1 = new CollectionType<string>();

IGereric<string> a;

obj1.Add("bye ");

obj1.Add("adyos ");

obj1.Add("sayonara ");

a = obj1;

a.Show();

CollectionType<Plant> plnt = new CollectionType<Plant>();

plnt.Add(new Rose());

plnt.Add(new Gladiolus());

plnt.Add(new Bush());

plnt.Show();

Console.WriteLine("\n");

CollectionType<Plant> obj2 = new CollectionType<Plant>();

ParseFile(obj2);

obj2.Show();

obj2.WriteFile(obj2);

}

}

}

LAB\_5.TXT WRITE.TXT

Rose

Rose

Cactus

Gladiolus

**ЛАБА 6**

CLASSES.CS

using Lab\_6.Exceptions;

using System;

using System.Diagnostics;

namespace Lab\_6

{

abstract class Plant

{

protected int numOfPlants = 0;

public string color { get; set; }

public int price { get; set; }

public override string ToString() => "Plant";

public Plant() { }

public virtual void Pour(int litters)

{

if (litters < 0)

throw new PlantException("Вы не можене полить растение < 0 литрами воды", litters);

Debug.Assert(litters < 50);

Console.WriteLine($"Вы полили растение {litters} литрами воды");

}

public abstract void ToPlant();

public void GetPlants() => Console.WriteLine($"Всего {numOfPlants} растеный вида {ToString()}");

}

sealed class Bush : Plant

{

public override string ToString() => "Bush";

public override void ToPlant() => Console.WriteLine("Вы посадили куст", numOfPlants += 1);

}

partial class Flower : Plant, IComparable

{

public override string ToString() => "Flower";

public override void ToPlant() => Console.WriteLine("Вы посадили цветок", numOfPlants += 1);

public override bool Equals(object obj)

{

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

return false;

Flower other = (Flower)obj;

return (numOfPlants == other.numOfPlants);

}

public override int GetHashCode()

{

return DateTime.Now.Millisecond;

}

public Flower(int Price, string Color)

{

if (Price < 0)

throw new FlowerException("Стоимость не может быть < 0", Price);

price = Price;

color = Color;

}

public int CompareTo(object obj)

{

Flower flower = (Flower)obj;

if (price > flower.price) return 1;

if (price < flower.price) return -1;

return 0;

}

}

class Rose : Plant, IPaper

{

public override string ToString() => "Rose";

public override void ToPlant() => Console.WriteLine("Вы посадили розу", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали букет роз");

public void PackIn() => Console.WriteLine("Вы обернули букет в бумагу");

}

class Gladiolus : Plant, IPaper

{

public override string ToString() => "Gladiolus";

public override void ToPlant() => Console.WriteLine("Вы посадили гладиолус", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали букет из гладиолусов");

public void PackIn() => Console.WriteLine("Вы обернули букет в бумагу");

}

class Cactus : Plant, IPot

{

public override string ToString() => "Cactus";

public override void ToPlant() => Console.WriteLine("Вы посадили кактус (abstract)", numOfPlants += 1);

void IPot.ToPlant() => Console.WriteLine("Вы посадили кактус (interface)", numOfPlants += 1);

public void Collect() => Console.WriteLine("Вы собрали кактус");

public void PutInPot() => Console.WriteLine("Вы поместили кактус в горшок");

}

}

CLASSES2.CS

using System;

namespace Lab\_6

{

struct FL

{

public string color;

public FL(string Color)

{

color = Color;

}

}

partial class Flower

{

private FL objFl;

private FL xFl = new FL("Red");

private FL xLFl = new FL("Green");

private FL xLLFl = new FL("Blue");

public enum enumFL

{

x = 1, xL, xLL

}

public void ChooseFL(enumFL num)

{

switch (num)

{

case enumFL.x: ///1

objFl = xFl;

Console.WriteLine("Вы выбрали Красный цветок");

break;

case enumFL.xL: ///2

objFl = xLFl;

Console.WriteLine("Вы выбрали Зеленый");

break;

case enumFL.xLL: ///3

objFl = xLLFl;

Console.WriteLine("Вы выбрали Синий");

break;

default:

Console.WriteLine("Такого цветка нет, пусть будет xFl");

objFl = xFl;

break;

}

}

}

}

INTERFACES.CS

namespace Lab\_6

{

public interface IPaper : IBouquet

{

void PackIn();

}

public interface IBouquet

{

void Collect();

}

public interface IPot : IBouquet

{

void PutInPot();

void ToPlant();

}

}

BOQUETTE.CS

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.IO;

namespace Lab\_6.Controller

{

class PriceComp : IComparable<Flower>

{

public int Compare(Flower x, Flower y)

{

if (x.price > y.price)

{

return 1;

}

else if (x.price < y.price)

{

return -1;

}

return 0;

}

public int CompareTo([AllowNull] Flower other)

{

throw new NotImplementedException();

}

}

class Bouquete

{

public List<Flower> Flwrs;

public Bouquete()

{

Flwrs = new List<Flower>();

}

public void Delete(int index)

{

Flwrs.RemoveAt(index);

}

public void Add(Flower item)

{

Flwrs.Add(item);

}

public int GetPrice()

{

int res = 0;

foreach (Flower item in Flwrs)

{

res += item.price;

}

return res;

}

}

class BouqContr

{

public static void FS(Bouquete b)

{

b.Flwrs.Sort();

}

public static void FindByColor(Bouquete b, string color)

{

foreach(Flower item in b.Flwrs)

{

if (item.color == color)

{

Console.WriteLine($"Цветок с цветом {color} был найден. Его цена: {item.price}");

}

}

}

public static void ParseFile(Bouquete bq1)

{

bool not\_initialized = true;

using (StreamReader stream = new StreamReader(@"K:\Uni\OOP-Labs-master\Lab\_6\Lab\_6\6\_lab.txt"))

{

while (stream.ReadLine() is string line)

{

switch (line)

{

case "#Flower" when not\_initialized:

Flower fl = new Flower(int.Parse(stream.ReadLine()), stream.ReadLine());

bq1.Add(fl);

break;

}

}

}

}

}

}

LOGGER.CS

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

using Lab\_6.Exceptions;

namespace Lab\_6.Logger

{

static class Logger

{

private static void Report(bool ifFile, string filePath, string log)

{

if (ifFile)

{

using (StreamWriter writer = new StreamWriter(filePath))

{

writer.Write(log);

}

}

else

Console.WriteLine(log);

}

public static void Writelog(CustomException ex, bool toFile = false, string filePath = @"K:\Uni\OOP-Labs-master\Lab\_6\Lab\_6\log.txt")

{

DateTime time = DateTime.Now;

string toLog = $"{time} INFO:\n" +

$"{ex.Message} \n{ex.ErrorClass}";

Report(toFile, filePath, toLog);

}

public class ConsoleLogger

{

public static void WriteLog(CustomException ex, string \_FilePath = @"K:\Uni\OOP-Labs-master\Lab\_6\Lab\_6\log.txt")

{

Logger.Writelog(ex, filePath: \_FilePath);

}

}

public class FileLogger

{

public static void WriteLog(CustomException ex, string \_FilePath = @"K:\Uni\OOP-Labs-master\Lab\_6\Lab\_6\log.txt")

{

Logger.Writelog(ex, true, filePath: \_FilePath);

}

}

}

}

FLOWEREXCEPTION.CS

namespace Lab\_6.Exceptions

{

class FlowerException : CustomException

{

public int ErrorPrice{ get; set; }

public FlowerException(string message, int errorPrice)

: base(message, "Flower")

{

this.ErrorPrice = errorPrice;

}

}

}

**RECTANGLE**

RECTANGLE.CS

using System;

namespace Podliva

{

interface Figure

{

string Print(Rectangle t);

}

[Serializable]

class Rectangle : Figure

{

public int x { get; set; }

public int y { get; set; }

public int l { get; set; }

public int h { get; set; }

public string color { get; set; }

public Rectangle() { }

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

{

this.x = x;

this.y = y;

this.color = color;

}

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

{

this.h = h;

this.l = l;

}

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

{

H.h = H.h + i;

H.h = H.l + i;

return H;

}

public int Sqr(Rectangle pl)

{

return pl.h \* pl.l;

}

public string Print(Rectangle t)

{

return t.x + " " + t.y + " " + t.l + " " + t.h + " " + t.color;

}

public string ToString()

{

return base.ToString() + " " + l + " " + y + " " + l;

}

}

}

PROGRAM.CS

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.Json;

using System.Threading.Tasks;

namespace Podliva

{

class Program

{

static async Task Main(string[] args)

{

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

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

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

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

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

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

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

listochek.Add(rec1);

listochek.Add(rec2);

listochek.Add(rec3);

listochek.Add(rec4);

listochek.Add(rec5);

listochek.Add(rec6);

rec2 = rec2 + 20;

Console.WriteLine(rec2.h + rec2.l);

/\*from i in listochek orderby i.h orderby i.l select i;\*/

var linq = listochek.OrderBy(t => t.h).OrderBy(t => t.l).Last();

//foreach (var item in linq)

//{

// Console.WriteLine(item.ToString());

//}

Console.WriteLine(linq.ToString());

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

{

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

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

}

}

}

}

**Transport, Air**

Абстрактный клас, наследовать его в другом классе. Свойства + перечисление. В интерфейсе 2 метода, он тож наследуется.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

/\*

Метод Check():

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

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

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

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

3. Сделать ВЕСЬ вывод еще и в файл.

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

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

5. Создать коллекцию из Air и добавить 5 объектов. С помощью linq запросов вывести количество самолетов, находящихся в Status = fly, а так же посчитать среднюю их скорость.

\*/

namespace Practice\_Exam

{

abstract class Transport

{

public string name;

}

interface IAirable

{

void Check();

void Fly();

}

interface IAirability

{

void Check();

}

class Air : Transport, IAirable, IAirability

{

public int CountOfPassengers;

public int Speed;

public enum Status { ready, fly, stop };

public Status st;

void IAirable.Check()

{

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

{

st = Status.stop;

sw.WriteLine("Status now {0}", st);

}

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

{

st = Status.ready;

sw.WriteLine("Status now {0}", st);

}

else

{

st = Status.fly;

sw.WriteLine("Status now {0}", st);

}

}

public void Fly()

{

try

{

if (st == Status.fly)

{

sw.WriteLine("FLYING");

}

else

{

throw new Exception();

}

}

catch (Exception e)

{

Console.WriteLine("{0} Exception caught.",e);

}

}

public void show()

{

string writePath = @"E:\ДОСДАТЬ\Practice\_Exam\Practice\_Exam\hta.txt";

using (FileStream fs = new FileStream(writePath, FileMode.Append))

{

using (StreamWriter sw = new StreamWriter(fs))

{

sw.WriteLine("All information about plane");

sw.WriteLine(this.name);

sw.WriteLine(this.CountOfPassengers);

sw.WriteLine(this.Speed);

sw.WriteLine(this.st);

}

}

}

}

class Program

{

static void Main(string[] args)

{

string writePath = @"E:\ДОСДАТЬ\Practice\_Exam\Practice\_Exam\hta.txt";

Air airplane1 = new Air();

airplane1.name = "BOING";

airplane1.CountOfPassengers = 21;

airplane1.Speed = 100;

airplane1.st = Air.Status.fly;

using (FileStream fs = new FileStream(writePath, FileMode.Append))

{

using (StreamWriter sw = new StreamWriter(fs))

{

sw.WriteLine(airplane1.name);

sw.WriteLine(airplane1.CountOfPassengers);

sw.WriteLine(airplane1.Speed);

sw.WriteLine(airplane1.st);

}

}

airplane1.show();

IAirability air1 = (IAirability)airplane1;

air1.Check();

IAirable air2 = (IAirable)airplane1;

air2.Check();

airplane1.Fly();

}

}

}

**USER**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Runtime.Serialization.Formatters.Binary;

namespace Practice\_Exam\_2\_Repited

{

[Serializable]

public class User : IComparable

{

private string email;

private int password;

public string status;

public string Status

{

set

{

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

{

status = value;

}

else

{

throw new Exception();

}

}

get

{

return status;

}

}

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

{

this.email = email;

this.password = password;

this.status = status;

}

public int CompareTo(object o)

{

User a = o as User;

if(a.password > this.password)

{

return 1;

}

else if(a.password < this.password)

{

return -1;

}

else

{

return 0;

}

}

public override string ToString()

{

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

}

public override int GetHashCode()

{

return (password - 235);

}

public override bool Equals(object obj)

{

User user1 = obj as User;

if (user1 != null)

{

if (user1.email == this.email)

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

}

[Serializable]

public class WedLet<T>

{

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

public void Add(T obj)

{

listic.AddLast(obj);

}

public void Remove(T obj)

{

listic.Remove(obj);

}

public int Count()

{

return listic.Count();

}

}

class Program

{

static void Main(string[] args)

{

try

{

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

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

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

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

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

Console.WriteLine(user2.ToString());

Console.WriteLine(user1.ToString());

Console.WriteLine(user3.ToString());

Console.WriteLine(user4.ToString());

Console.WriteLine(user5.ToString());

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

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

github.Add(user1);

github.Add(user2);

github.Add(user3);

github.Add(user4);

github.Add(user5);

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

where s.status == "signin"

select s;

Console.WriteLine(linq1.Count());

var serialize = new BinaryFormatter();

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

{

serialize.Serialize(file, github);

}

}

catch (Exception e)

{

Console.WriteLine(e);

}

}

}

}

**LOCATION TAXI PARK**

using System;

using System.Collections.Generic;

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));

}

}

}

**SOMESTRING**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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);

}

}

}

}

**BILL**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Xml;

using System.Xml.Serialization;

namespace Practice\_Exam\_3

{

class Program

{

interface INumber

{

int Number { get; set; }

}

[Serializable]

public class Bill : INumber

{

public int num;

public Bill(int number)

{

this.Number = number;

}

public int Number

{

get

{

return num;

}

set

{

if ((value != 5 && value != 10 && value != 20 && value != 100) || value < 0)

{

throw new Error();

}

else

{

num = value;

}

}

}

public Bill()

{

}

public override string ToString()

{

return base.ToString();

}

}

[Serializable]

public class Wallet<T> where T : Bill

{

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

public void Add(T obj)

{

if (money.Count > 200)

{

throw new ToMuchMoney();

}

else

{

money.Add(obj);

}

}

public void Remove()

{

if (money.Count > 0)

{

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

money.Remove(delete.First());

}

else

{

throw new NoBillWallet();

}

}

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()}");

}

}

}

class ToMuchMoney : Exception

{

public ToMuchMoney() : base("To much money") { }

}

class NoBillWallet : Exception

{

public NoBillWallet() : base("No Bill in Wallet") { }

}

class Error : Exception

{

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

}

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);

}

wallet.Count();

wallet.Remove();

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

wallet.Count();

Console.WriteLine(wallet.ToString());

}

catch (ToMuchMoney err)

{

Console.WriteLine(err.Message);

}

catch (NoBillWallet err)

{

Console.WriteLine(err.Message);

}

catch (Error err)

{

Console.WriteLine(err.Message);

}

}

}

}

**BUTTON**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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);

}

}

}

}

ОБОБЩЕННАЯ КОЛЛЕКЦИЯ ТИПА ОЧЕРЕДЬ

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();

}

АБСТРАКТНЫЙ КЛАСС + ВИРТУАЛЬНАЯ ФУНКЦИЯ, УРАВНЕНИЯ

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 Liner : Function

{

public double A { get; set; }

public double B { get; set; }

}

class Sqr : Function

{

public int C { get; set; }

}

public delegate string MyDelegate();

class ArrayFunc<t>

{

public static string ToString()

{

return "Work";

}

new public static string GetHashCode()

{

return "Work";

}

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

}

ИНТЕРФЕЙС, СВОЙСТВО И МЕТОДЫ

interface IScore

{

int Amount { get; set; }

int AddSum(int money);

int RemSum(int money);

}

abstract class Human

{

public string DateofBirth { get; set; }

}

class Person : Human, IScore

{

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

public int AddSum(int money)

{

return Amount += money;

}

public int RemSum(int money)

{

return Amount -= money;

}

static int count;

public Person(string date){

DateofBirth = date;

count++;

}

static Person()

{

Console.WriteLine("Статический конструктор");

}

public static int CountObj()

{

return count;

}

public override bool Equals(object obj)

{

if (obj == null) return false;

Person pers = (Person)obj;

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

}

}

ИНТЕРФЕЙС:

interface IDo {

void Go();

int Jump(int a);

int Energy {get; } //шаблон свойства

}

interface IKnow {

Void Count();

int Math();

}

interface IPosebel : IDo, IKnow { }

interface IComparable

{

int CompareTo(object obj)

}

System.Collections

interface IComparer

{

int Compare(object obj1,object obj2)

}

ДЕЛЕГАТ

delegate int **IntOperation** (int i, int j);

class Program {

        static int Sum(int x, int y) { return x + y; }

        static int Mul(int x, int y) { return x \* y; }

        static void Main() {

**IntOperation** op1 = new **IntOperation**(Sum);

            int result = op1(5, 10);

            Console.WriteLine("Сумма: " + result);

            op1 = new **IntOperation**(Mul); // меняем ссылку

            result = op1(5, 10);

            Console.WriteLine("Произведение: " + result);

        }

}

ОБОБЩЕННЫЙ ДЕЛЕГАТ:

delegate T MyDel<T> (T obj1, T obj2); // обобщ. дел.

class MySum {

        public static int SumInt(int a, int b) {return a + b; }

        public static string SumStr(string s1, string s2) { return s1 + " " + s2;}

        public static char SumCh(char a, char b){ return (char)(a + b); }

}

class Program {

        static void Main() {

         MyDel<int> del1 = MySum.SumInt;

         Console.WriteLine("6 + 7 = " + del1(6,7)); //вывод: 6 + 7 = 13

         MyDel<char> del3 = MySum.SumCh;

         Console.WriteLine("'a' + 'c' = " + del3('a','c')); //вывод: ‘а’ + ‘с’ = A

        }

}

**Типы коллекций:**

* необобщенные (наличие разнотипных данных, ссылки на данные типа object (не обеспечивают типовую безопасность), System.Collections)
* Обобщенные (обеспечивают типовую безопасность, System.Collections.Generic)
* специальные (System.Collections.Specialized)
* с поразрядной организацией (BitArray)
* Параллельные (многопоточный доступ к коллекции, System.Collections.Concurrent)

**Необобщенные коллекции** (ArrayList (IList, ICollection, IEnumerable, ICloneabl), BitArray (ICollection, IEnumerable, ICloneable), Hashtable, Queue, SortedList (класс коллекции, хранящей наборы пар "ключ-значение", отсортированных по ключу), Stack).

**Класс Directory (СТАТИЧЕСКИЙ)**

Класс Directory предоставляет ряд статических методов для управления каталогами. Некоторые из этих методов:

* CreateDirectory(path) - создает каталог по указанному пути path
* Delete(path) - удаляет каталог по указанному пути path
* Exists(path) - определяет, существует ли каталог по указанному пути path. Если существует, возвращается true, если не существует, то false
* GetDirectories(path): получает список каталогов в каталоге path
* GetFiles(path): получает список файлов в каталоге path
* Move(sourceDirName, destDirName): перемещает каталог
* GetParent(path): получение родительского каталога

**Класс DirectoryInfo**

Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory. Некоторые из его свойств и методов:

* Create(): создает каталог
* CreateSubdirectory(path): создает подкаталог по указанному пути path
* Delete(): удаляет каталог
* Свойство Exists: определяет, существует ли каталог
* GetDirectories(): получает список каталогов
* GetFiles(): получает список файлов
* MoveTo(destDirName): перемещает каталог
* Свойство Parent: получение родительского каталога
* Свойство Root: получение корневого каталога

Создание каталога:

string path = @"C:\SomeDir";

string subpath = @"program\avalon";

DirectoryInfo dirInfo = new DirectoryInfo(path);

if (!dirInfo.Exists)

{ dirInfo.Create(); }

dirInfo.CreateSubdirectory(subpath);

Методы и свойства класса **FileInfo**:

* CopyTo(path): копирует файл в новое место по указанному пути path
* Create(): создает файл
* Delete(): удаляет файл --- объект\_файла.Delete()
* MoveTo(destFileName): перемещает файл в новое место
* Свойство Directory: получает родительский каталог в виде объекта DirectoryInfo
* Свойство DirectoryName: получает полный путь к родительскому каталогу
* Свойство Exists: указывает, существует ли файл
* Свойство Length: получает размер файла
* Свойство Extension: получает расширение файла
* Свойство Name: получает имя файла
* Свойство FullName: получает полное имя файла

Класс **File**: (СТАТИЧЕСКИЙ) - т.е. не привязан к конкретному файлу:

* Copy(): копирует файл в новое место - File.Copy(path)
* Create(): создает файл
* Delete(): удаляет файл --- File.Delete(path)
* Move: перемещает файл в новое место
* Exists(file): определяет, существует ли файл

Получение информации о файле

string path = @"C:\apache\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

    Console.WriteLine("Имя файла: {0}", fileInf.Name);

    Console.WriteLine("Время создания: {0}", fileInf.CreationTime);

    Console.WriteLine("Размер: {0}", fileInf.Length);

}