**LABA 2**

using System;

using System.Linq;

namespace laba2

{

class Program

{

static void Main(string[] args)

{

// ТИПЫ //

//Определение переменных

Console.WriteLine($"Введите переменную типа bool: ");

string bufer = Console.ReadLine();

bool q = Convert.ToBoolean(bufer);

byte w = 1;

sbyte e = 1;

char r = 'r';

decimal t = 12.3m;

double y = 11.01;

float u = 12.3f;

int i = -1;

uint o = 815;

long p = -100;

ulong a = 904;

short s = -7;

ushort d = 4;

Console.WriteLine($"bool: {q} \n" +

$"byte: {w} \n" +

$"sbyte: {e} \n" +

$"char: {r} \n" +

$"decimal: {t} \n" +

$"double: {y} \n" +

$"float: {u} \n" +

$"int: {i} \n" +

$"uint: {o} \n" +

$"long: {p} \n" +

$"ulong: {a} \n" +

$"short: {s} \n" +

$"ushort: {d} \n");

//Явные преобразования

int f;

float g;

y = (int)i;

Console.WriteLine($"{y}");

u = (int)i;

Console.WriteLine($"{u}");

t = (int)i;

Console.WriteLine($"{t}");

f = (int)i;

Console.WriteLine($"{f}");

g = (int)i;

Console.WriteLine($"{g}");

//Неявные преобразования

int num = -1;

long h = num;

Console.WriteLine($"{h}");

int j = num;

Console.WriteLine($"{j}");

double k = num;

Console.WriteLine($"{k}");

float l = num;

Console.WriteLine($"{l}");

double z = num;

Console.WriteLine($"{z}");

//упаковка и распаковка

int ii = 123;

object v = ii;

int m = (int)o;

//работа с неявно типизированной переменной

var qq = 15;

var ww = "Dada ya da";

//nullable

int? qwe;

//Определить переменную var

//var rrк = 5;

//rrк = 5.1;

//Ошибка из-за того, что неявной переменной присвоили значение int,

//а потом ей же присвоили значение float.

////////////////////////////////////////////////////////////////////

// СТРОКИ //

//Объявите строковые литералы, сравните их

var string1 = "Я съел пюре из брюквы";

var string2 = "Я съел луковицу";

var equal = string1 == string2;

Console.WriteLine(equal);

//Сцепление, копирование и тд.

var string11 = "Я";

var string12 = "съел";

var string13 = "пюре";

string string22 = string11 + ' ' + string12 + ' ' + string13;

Console.WriteLine(string22);

//Копирование

string11 = String.Copy(string12);

//Разделение на слова

string[] info = { "Name: Alex", "Dada", "Eto ya" };

Console.WriteLine(info);

int found = 0;

//Удаление

foreach (string ss in info)

{

found = ss.IndexOf(":");

Console.WriteLine("{0}", ss.Substring(found + 2));

}

//IsNullOrEmpty

string st1 = " ";

string st2 = null;

string[] array = { st1, st2 };

if (String.IsNullOrEmpty(st1))

{

Console.WriteLine("Is Null");

}

else { Console.WriteLine("Is Empty"); }

if (String.IsNullOrEmpty(st2))

{

Console.WriteLine("Is Null");

}

else { Console.WriteLine("Is Empty"); }

//String Builder

System.Text.StringBuilder sb = new System.Text.StringBuilder("съел пюре ");

sb.Insert(0, "Я ");

sb.AppendFormat("из брюквы");

Console.WriteLine(sb);

// МАССИВЫ //

//a

int[,] myArray = { { 0, 1, 2 }, { 3, 4, 5 } };

int rows = myArray.GetUpperBound(0) + 1;

int columns = myArray.Length / rows;

for (int count = 0; count < rows; count++)

{

for (int counter = 0; counter < columns; counter++)

{

Console.Write($"{myArray[count, counter]} \t");

}

Console.WriteLine();

}

//b

string[] daysOfWeek = { "Monday", "Tuersday", "Wednesday", "Thirsday", "Friday", "Saturday", "Sunday" };

for (int jj = 0; jj < daysOfWeek.Length; jj++)

{

Console.WriteLine(daysOfWeek[jj]);

Console.WriteLine(daysOfWeek.Length);

}

// c

Random rand = new Random();

int rr;

int[][] array3 = new int[3][];

array3[0] = new int[2];

array3[1] = new int[3];

array3[2] = new int[4];

for (rr = 0; r < 2; r++)

{

array3[0][r] = rand.Next(1, 15);

Console.WriteLine("{0}", array3[0][r]);

}

Console.WriteLine();

for (rr = 0; r < 3; r++)

{

array3[1][r] = rand.Next(1, 15);

Console.WriteLine("{0}", array3[1][r]);

}

Console.WriteLine();

for (rr = 0; r < 4; r++)

{

array3[2][r] = rand.Next(1, 15);

Console.WriteLine("{0}", array3[2][r]);

}

Console.WriteLine();

//d

var array4 = new Object[0];

var str1 = "";

// КОРТЕЖИ //

//a

(int, string, char, string, ulong) cart1 = (7, "Hi", 'r', "Brukva", 14);

//b

Console.WriteLine(cart1);

Console.WriteLine(cart1.Item1 + " " + cart1.Item4);

//c

(var aa, var bb) = (10, "брюкв");

Console.WriteLine(aa + " " + bb);

//d

var cart3 = (A: 1, B: 2);

var cart4 = (A: 1, B: 2);

Console.WriteLine((cart3.Item1, cart3.Item2) == (cart4.Item1, cart4.Item2));

// Создайте локальную функцию в main и вызовите ее //

static (int, int, int, char) fun(int[] a, string b)

{

int max = a.Max<int>();

int min = a.Min<int>();

int sum = a.Sum();

(int, int, int, char) kor = (max, min, sum, b[0]);

Console.WriteLine(kor);

return kor;

}

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

fun(arr, "abcdefg");

//Работа с checked/unchecked

int function1()

{

int maxVal = 2147483647;

int z = 0;

try

{

z = checked(maxVal + 11);

}

catch (System.OverflowException e)

{

Console.WriteLine(e.ToString());

}

return z;

}

int function2()

{

int maxVal = 2147483647;

int z = 0;

try

{

z = maxVal + 11;

}

catch (System.OverflowException e)

{

Console.WriteLine(e.ToString());

}

return z;

}

Console.WriteLine("fisrt : {0}", function1());

Console.WriteLine("second : {0}", function2());

}

}

}

**LABA 3**

using System;

namespace laba3

{

class Program

{

public partial class Book

{

//Свойства get set

private int o = 49;

public int a

{

get

{

return o;

}

protected set

{

o = value;

}

}

public int idP;

public readonly string nameP;

public string authorP;

public string publishP;

public const int pages\_pu = 228;

public int costP;

public string bindtypeP;

public static int idS;

public static readonly string nameS;

public static string authorS;

public static string publishS;

public static int pagesS;

public static int costS;

public static string bindtypeS;

static int count = 0;

static Book()

{

idS = 112233;

nameS = "Война и Мир";

authorS = "Булгаков";

publishS = "2012";

pagesS = 1430;

costS = 25;

bindtypeS = "Твердый";

count++;

}

public Book(ref string x)

{

idP = 123123;

nameP = x;

authorP = "Лермонтов";

publishP = "2011";

costP = 25;

bindtypeP = "Твердый";

count++;

}

private Book(string y)

{

idP = 123123;

nameP = "Война и Мир";

authorP = "Толстой";

publishP = y;

costP = 25;

bindtypeP = "Твердый";

count++;

}

public Book(string z = "Горе от ума", string v = "мягкий")

{

nameP = z;

authorP = "Пушкин";

publishP = "2010";

costP = 25;

bindtypeP = v;

count++;

}

public static void DisplayCounter()

{

Console.WriteLine($"Создано {count} объектов ");

}

public void GetInfo1()

{

Console.WriteLine($"-----------------------------------\n" +

$"ID книги:{idP} \n" +

$" Название книги:{nameP}\n" +

$" Автор:{authorP}\n" +

$" Издательство:{publishP}\n" +

$" Кол-во страниц:{pages\_pu}" +

$"Стоимость:{costP}\n" +

$" Тип переплета:{bindtypeP}" +

$"\n-----------------------------------\n\n\n");

}

public void GetInfo2()

{

Console.WriteLine($"-----------------------------------\n" +

$"ID книги:{idS} \n" +

$" Название книги:{nameS}\n" +

$" Автор:{authorS}\n" +

$" Издательство:{publishS}\n" +

$" Кол-во страниц:{pagesS}" +

$"Стоимость:{costS}\n" +

$" Тип переплета:{bindtypeS}" +

$"\n-----------------------------------\n\n\n");

}

public override string ToString()

{

return "Переопределенный тустринг: " + costS;

}

}

static void Main(string[] args)

{

string b = "Книжка";

Book Book1 = new Book();

Book Book2 = new Book(ref b);

Book Book3 = new Book("название чего-нибудь");

Book1.GetInfo2();

Book2.GetInfo1();

Book3.GetInfo1();

Book.DisplayCounter();

Book c1 = new Book();

Book c2 = new Book();

Console.WriteLine("Tostring::{0}", c1.ToString());

Console.WriteLine("Tostring::{0}", c2.ToString());

Console.WriteLine("\n\n\nHashcode1: {0}", c1.GetHashCode());

Console.WriteLine("Hashcode1: {0}", c2.GetHashCode());

if (c1.Equals(c2) && c2.Equals(c1))

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

else

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

Console.WriteLine("Введите автора");

string u = Console.ReadLine();

Console.WriteLine("Введите год издания");

string k = Console.ReadLine();

Book[] arr = new Book[] { Book2, Book3 };

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

{

if (u == arr[i].authorP && k == arr[i].publishP)

{

Console.WriteLine($"-----------------------------------\n" +

$"ID книги:{arr[i].idP} \n" +

$" Название книги:{arr[i].nameP}\n" +

$" Автор:{arr[i].authorP}\n" +

$" Издательство:{arr[i].publishP}\n" +

$"Стоимость:{arr[i].costP}\n" +

$"Тип переплета:{arr[i].bindtypeP}\n" +

$"\n-----------------------------------\n\n\n");

}

else

{

if (u != arr[1].authorP

&& u != arr[0].authorP

|| k != arr[0].publishP

&& k != arr[1].publishP)

{

Console.WriteLine("В заданной конфигурации книг нет");

break;

}

else

continue;

}

}

var anonim = new { Name = "Книга", value = 5 };

Console.WriteLine($"{anonim}");

}

}

}

**LABA 4**

using System;

using System.Collections.Generic;

namespace laba4

{

class MyList<T> : List<T>

{

Owner Alex = new Owner(4409, "Alex", "MyCorp");

Date date = new Date(16, 10, 2020);

public static MyList<T> operator +(MyList<T> a, T b)

{

a.Add(b);

return a;

}

public static MyList<T> operator -(MyList<T> a, T b)

{

a.Remove(b);

return a;

}

public static bool operator !=(MyList<T> list1, MyList<T> list2)

{

if (list1.Count != list2.Count)

{

return true;

}

for (var i = 0; i < list1.Count; i++)

{

if (!object.Equals(list1[i], list2[i]))

{

return true;

}

}

return false;

}

public static bool operator ==(MyList<T> list1, MyList<T> list2)

{

if (list1.Count != list2.Count)

{

return false;

}

for (var i = 0; i < list1.Count; i++)

{

if (!object.Equals(list1[i], list2[i]))

{

return false;

}

}

return true;

}

public override string ToString()

{

var str = string.Empty;

foreach (var item in this)

{

str = str + item + " ";

}

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

return str;

}

class Date

{

private int day;

private int month;

private int year;

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

{

day = a;

month = b;

year = c;

}

}

}

class Owner

{

private int id;

private string name;

private string name\_corossion;

public Owner(int a, string b, string c)

{

id = a;

name = b;

name\_corossion = c;

}

}

public static class Extensions

{

public static int CalculateItems<T>(this List<T> list)

{

return list.Count;

}

}

public static class NullElements

{

public static bool SearchNullElements<T>(this List<T> list)

{

bool a;

a = list == null;

return a;

}

}

public static class Sum

{

public static string SearchSum<T>(this List<T> list)

{

string s3 = list[0] + " " + list[1];

return s3;

}

}

class Program

{

static void Main(string[] args)

{

var list = new MyList<string>();

var list1 = new MyList<string>();

Console.WriteLine($"Введите первый элемент");

string el1 = Console.ReadLine();

Console.WriteLine($"Введите второй элемент");

string el2 = Console.ReadLine();

Console.WriteLine($"Введите третий элемент");

string el3 = Console.ReadLine();

Console.WriteLine($"Введите четвертый элемент");

string el4 = Console.ReadLine();

list = list + el1;

list = list + el2;

list = list + el3;

list = list + el4;

list = list1 + el1;

list = list1 + el2;

list = list1 + el3;

list = list1 + el4;

Console.WriteLine(list);

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

list = list - el4;

bool result = list != list1;

Console.WriteLine(list);

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

var itmesCount = list.CalculateItems();

var nooelem = list.SearchNullElements();

var sum = list.SearchSum();

Console.WriteLine($"Подсчет количества слов: {itmesCount};");

Console.WriteLine($"Проверка списка на нулевые элементы: {nooelem};");

Console.WriteLine($"Проверка на неравенство: {result};");

Console.WriteLine($"Сложение строк: {sum};");

}

}

}

**LABA 5**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab5

{

class Program

{

static void Main(string[] args)

{

Hunter hunter1 = new Hunter("Sam", 1743, 190, "kosa", 24);

Archer archer1 = new Archer("Gos", 7312, 185, "lyk", 32, 1000);

Shaman shaman1 = new Shaman("Trall", 3523, 215, "magic", 25);

Psychic psychic1 = new Psychic("Sun", 4823, 170, "golos", 39, 1000);

psychic1.Clone(); // вызов метода у класса

((Ialive)psychic1).Clone(); // вызов метода у интерфейса

Console.WriteLine("hunter1 is Hunter:" + (hunter1 is Hunter));

Console.WriteLine("hunter1 is Archer:" + (hunter1 is Archer));

Console.WriteLine("archer1 is Hunter:" + (archer1 is Hunter));

Console.WriteLine("archer1 is Archer:" + (archer1 is Archer));

//ToString дял всех

Console.WriteLine(archer1.ToString());

// printer

Printer printer = new Printer();

printer.IAmPrinting(psychic1);

Console.WriteLine("------------------Вывод через массив------------------------------");

Fighter[] fighters = new Fighter[] { hunter1, archer1, shaman1, psychic1 };

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

{

printer.IAmPrinting(fighters[i]);

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

}

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

GenFighter<Fighter>.Func(shaman1);

}

}

}

////////////////////////////////

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab5

{

interface Iactionable

{

void Run();

void Attack();

void Protect();

void Speek();

}

interface Ialive

{

void Clone();

}

public abstract class Fighter : Iactionable

{

protected string name;

protected int age;

protected int height;

protected string weapon;

protected int id;

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

public int Height

{

get { return height; }

set { height = value; }

}

public string Weapon

{

get { return weapon; }

set { weapon = value; }

}

public int Id

{

get { return id; }

set { id = value; }

}

public override string ToString() // переопределение с выводом доп инфы

{

Console.WriteLine(Name + " " + Age + " " + Height + " " + Weapon + " " + Id);

return " type " + base.ToString();

}

virtual public bool ChangeWeapon(string newWeapon)

{

return true;

}

public abstract void Run(); // откладываем реализацию методов

public abstract void Attack();

public abstract void Protect();

public abstract void Speek();

public abstract void Clone();

}

public class Hunter : Fighter, Ialive

{

public Hunter(string name, int id, int height, string weapon, int age)

{

this.name = name;

this.id = id;

this.height = height;

this.weapon = weapon;

this.age = age;

}

override public void Speek() // реализуем методы из интерфейса

{

Console.WriteLine("Hello");

}

public override void Attack()

{

Console.WriteLine("Атака завершена");

}

public override void Run()

{

Console.WriteLine("Бежать вперёд");

}

public override void Protect()

{

Console.WriteLine("Защищаться");

}

public override void Clone() // метод из абстрактного класса с таким же названием

{

Console.WriteLine("Это метод из астрактного класса");

}

void Ialive.Clone() // метод из интерсфеса с таким же названием

{

Console.WriteLine("метод интерфейса");

}

public override bool ChangeWeapon(string newWeapon)

{

if (newWeapon != null && newWeapon != "")

{

name = newWeapon;

return true;

}

else return false;

}

}

public class Shaman : Fighter, Ialive

{

public Shaman(string name, int id, int height, string weapon, int age)

{

this.name = name;

this.id = id;

this.height = height;

this.weapon = weapon;

this.age = age;

}

public override void Clone()

{

Console.WriteLine("Это метод из астрактного класса");

}

void Ialive.Clone()

{

Console.WriteLine("метод интерфейса");

}

override public void Speek()

{

Console.WriteLine("Hello");

}

public override void Attack()

{

Console.WriteLine("Атака завершена");

}

public override void Run()

{

Console.WriteLine("Бежать вперёд");

}

public override void Protect()

{

Console.WriteLine("Защищаться");

}

public override bool ChangeWeapon(string newWeapon)

{

if (newWeapon != null && newWeapon != "")

{

name = newWeapon;

return true;

}

else return false;

}

}

sealed public class Psychic : Shaman

{

public int soulsNumber;

public Psychic(string name, int id, int height, string weapon, int age, int soulsNumber) : base(name, id, height, weapon, age)

{

this.soulsNumber = soulsNumber;

}

public bool canAttack()

{

if (soulsNumber < 7)

return false;

else

return true;

}

}

sealed public class Archer : Hunter

{

int arrowsNumber;

public Archer(string name, int id, int height, string weapon, int age, int arrowsNumber) : base(name, id, height, weapon, age)

{

this.arrowsNumber = arrowsNumber;

}

public bool useTenArrows()

{

if (arrowsNumber < 10)

return false;

else

return true;

}

}

public static class GenFighter<Type> where Type : Fighter

{

public static void Func(Type fighter)

{

Console.WriteLine($"Имя {fighter.Name}");

Console.WriteLine($"Возраст {fighter.Age}");

Console.WriteLine($"Возраст {fighter.Weapon}");

}

}

public class forObject : Object // переопределение методов Object

{

public override bool Equals(object obj)

{

if (obj == null) return false;

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

return true; ;

}

public string Name { get; set; }

int sNumber;

public override int GetHashCode()

{

int hash = 269;

hash = string.IsNullOrEmpty(Name) ? 0 : Name.GetHashCode();

hash = (hash \* 47) + sNumber.GetHashCode();

return hash;

}

}

public class Printer

{

public void IAmPrinting(Fighter fighter)

{

Console.WriteLine(fighter.ToString());

}

}

}

**LABA 6**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba6

{

enum Doc : int

{

Receipt, Waybill, Check

}

partial struct Information

{

public string client;

public string organization;

}

interface IDocument

{

void Info();

}

public abstract class Document

{

private string title;

public string Title

{

get { return title; }

set { title = value; }

}

private DateTime date;

public DateTime Date

{

get { return date; }

set { date = value; }

}

private Client client;

private Organization organization;

public Document(string title, DateTime date, Client client, Organization organization)

{

this.title = title;

this.date = date;

this.client = client;

this.organization = organization;

}

public string Name

{

get { return client.Name; }

set { client.Name = value; }

}

public string NameOfOrganization

{

get { return organization.NameOfOrganization; }

set { organization.NameOfOrganization = value; }

}

public abstract void Info();

virtual public int GetTotalPrice() { return 0; }

}

sealed public class Receipt : Document, IDocument //квитанция

{

private int servicePrice;

public Receipt(string title, DateTime date, Client client, Organization organization, int servicePrice) : base(title, date, client, organization)

{

this.servicePrice = servicePrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + servicePrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + servicePrice);

}

override public int GetTotalPrice()

{

return servicePrice;

}

}

sealed public class Waybill : Document, IDocument //накладная

{

private int servicePrice;

public Waybill(string title, DateTime date, Client client, Organization organization, int servicePrice) : base(title, date, client, organization)

{

this.servicePrice = servicePrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + servicePrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + servicePrice);

}

override public int GetTotalPrice()

{

return servicePrice;

}

}

sealed public class Check : Document, IDocument

{

private int totalPrice;

public Check(string title, DateTime date, Client client, Organization organization, int totalPrice) : base(title, date, client, organization)

{

this.totalPrice = totalPrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + totalPrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + totalPrice);

}

override public int GetTotalPrice()

{

return totalPrice;

}

}

public class Client

{

private string name;

public Client(string name)

{

this.name = name;

}

public string Name

{

get { return name; }

set { name = value; }

}

}

public class Organization

{

private string nameOfOrganization;

public Organization(string nameOfOrganization)

{

this.nameOfOrganization = nameOfOrganization;

}

public string NameOfOrganization

{

get { return nameOfOrganization; }

set { nameOfOrganization = value; }

}

}

public class Summa

{

List<Receipt> receipts = new List<Receipt>();

List<Waybill> waybills = new List<Waybill>();

List<Check> checks = new List<Check>();

public Receipt this[int ind]

{

get

{

if (ind > receipts.Count)

{

Console.WriteLine("Превышен максимальный индекс списка квитанций"); return null;

}

return receipts[ind];

}

set

{

if (ind > receipts.Count)

Console.WriteLine("Элемента с таким индексом не существует");

else receipts[ind] = value;

}

}

public void AddReceipt(Receipt a) { receipts.Add(a); }

public void AddWaybill(Waybill a) { waybills.Add(a); }

public void AddCheck(Check a) { checks.Add(a); }

public void DelReceipt(Receipt a) { receipts.Remove(a); }

public void DelWaybill(Waybill a) { waybills.Remove(a); }

public void DelCheck(Check a) { checks.Remove(a); }

public void ShowList()

{

Console.WriteLine("\n" + "Список квитанций: ");

foreach (Receipt item in receipts)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine("\n" + "Список накладных: ");

foreach (Waybill item in waybills)

{

Console.WriteLine(item.ToString());

}

Console.WriteLine("\n" + "Список чеков: ");

foreach (Check item in checks)

{

Console.WriteLine(item.ToString());

}

}

public int GetWaybillPrice(string name)

{

int price = 0;

foreach (Waybill item in waybills)

{

if (item.Title == name)

price += item.GetTotalPrice();

}

return price;

}

public int GetCheckCount()

{

Console.WriteLine("");

Console.WriteLine("Количество чеков:");

return checks.Count;

}

public void GetTime(DateTime firstDate, DateTime secondDate)

{

Console.WriteLine("");

foreach (Receipt item in receipts)

if (item.Date > firstDate && item.Date < secondDate)

Console.WriteLine(item.ToString());

foreach (Waybill item in waybills)

if (item.Date > firstDate && item.Date < secondDate)

Console.WriteLine(item.ToString());

foreach (Check item in checks)

if (item.Date > firstDate && item.Date < secondDate)

Console.WriteLine(item.ToString());

}

public class SummaController

{

public void Show(Summa lis) { lis.ShowList(); }

public int Price(Summa lis, string name) { int a = lis.GetWaybillPrice(name); return a; }

public int Count(Summa lis) { int a = lis.GetCheckCount(); return a; }

public void Get(Summa lis, DateTime firstDate, DateTime lastDate) { lis.GetTime(firstDate, lastDate); }

}

}

partial struct Information

{

public string document;

}

class Program

{

static void Main(string[] args)

{

Client first\_client = new Client("Самцевич Алексей");

Waybill first\_waybill = new Waybill("Накладная 1", new DateTime(2020, 11, 05), first\_client, new Organization("111"), 7800);

Receipt first\_receipt = new Receipt("Квитанция 1", new DateTime(2020, 11, 05), first\_client, new Organization("222"), 650);

Check first\_check = new Check("Чек 1", new DateTime(2019, 11, 05), first\_client, new Organization("333"), 150);

Check second\_check = new Check("Чек 2", new DateTime(2019, 11, 05), first\_client, new Organization("444"), 10);

Waybill second\_waybill = new Waybill("Накладная 2", new DateTime(2016, 06, 12), first\_client, new Organization("555"), 1200);

Waybill third\_waybill = new Waybill("Накладная 3", new DateTime(2015, 04, 01), first\_client, new Organization("666"), 800);

Waybill fours\_waybill = new Waybill("Накладная 4", new DateTime(2020, 07, 12), first\_client, new Organization("777"), 1200);

Summa summa = new Summa();

summa.AddWaybill(first\_waybill);

summa.AddReceipt(first\_receipt);

summa.AddCheck(first\_check);

summa.AddCheck(second\_check);

summa.AddWaybill(second\_waybill);

summa.AddWaybill(third\_waybill);

summa.AddWaybill(fours\_waybill);

summa.ShowList();

Console.WriteLine("\n" + "Суммарную стоимость продукции заданного наименования по всем накладным = {0}", summa.GetWaybillPrice("Накладная 1"));

Console.WriteLine(summa.GetCheckCount());

summa.GetTime(new DateTime(2019, 01, 01), new DateTime(2021, 01, 01));

//Console.WriteLine(first\_waybill);

}

}

}

**LABA 7**

//////////////////////////////////////////

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

namespace laba7

{

class Program

{

static void Main(string[] args)

{

Client first\_client = new Client("Самцевич Алексей");

try

{

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

try

{

Waybill first\_waybill = new Waybill("Накладная 1", new DateTime(2020, 11, 05), first\_client, new Organization("111"), 7800, 0);

}

catch (IsNotNameOfOrganization ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

try

{

Receipt first\_receipt = new Receipt("Накладная 2", new DateTime(2020, 11, 05), first\_client, new Organization("222"), 7800, 1);

}

catch (IsNotTitle ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

try

{

Check first\_check = new Check("Накладная 3", new DateTime(2020, 11, 05), first\_client, new Organization("333"), 7800, 1);

}

catch (WrongIdValue ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

Waybill second\_waybill = new Waybill("Накладная 1", new DateTime(2020, 11, 05), first\_client, new Organization("111"), 7800, 0);

Waybill third\_waybill = new Waybill("Накладная 2", new DateTime(2020, 11, 05), first\_client, new Organization("222"), 7800, 1);

Waybill fours\_waybill = new Waybill("Накладная 3", new DateTime(2020, 1я1, 05), first\_client, new Organization("333"), 7800, 1);

Document[] documents = new Document[] { second\_waybill, third\_waybill, fours\_waybill };

try

{

Console.WriteLine(documents[4].Id);

}

catch (IndexOutOfRangeException ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

try

{

object obj = third\_waybill.Title;

int title = (int)obj;

}

catch (InvalidCastException ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

try

{

double countHashCode = third\_waybill.Id / second\_waybill.Id;

}

catch (DivideByZeroException ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

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

int index = 10;

Debug.Assert(index > -1, "Значение должно быть больше -1");

}

catch (Exception ex)

{

Console.WriteLine($"{ex.Message}\n{ex.Source}\n{ex.StackTrace}");

}

finally

{

Console.WriteLine("Finally block");

}

}

}

}

//////////////////////

using System;

using System.Collections.Generic;

namespace laba7

{

interface IDocument

{

void Info();

}

public abstract class Document

{

private string title;

public string Title

{

get { return title; }

set { title = value; }

}

private DateTime date;

public DateTime Date

{

get { return date; }

set { date = value; }

}

public int id;

public int Id

{

get { return id; }

set

{

if (value < 0)

{

throw new WrongIdValue("Недопустимое значение для задания id ", value);

}

else id= value ;

}

}

private Client client;

private Organization organization;

public Document(string title, DateTime date, Client client, Organization organization, int id)

{

if (title.Length <= 3)

{

throw new IsNotTitle("Недопустимое значение для названия документа", title);

}

else this.title = title;

this.date = date;

this.client = client;

this.organization = organization;

this.id = id;

}

public string Name

{

get { return client.Name; }

set { client.Name = value; }

}

public string NameOfOrganization

{

get { return organization.NameOfOrganization; }

set { organization.NameOfOrganization = value; }

}

public abstract void Info();

virtual public int GetTotalPrice() { return 0; }

}

sealed public class Receipt : Document, IDocument //квитанция

{

private int servicePrice;

public Receipt(string title, DateTime date, Client client, Organization organization, int servicePrice, int id) : base(title, date, client, organization, id)

{

this.servicePrice = servicePrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + servicePrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + servicePrice);

}

override public int GetTotalPrice()

{

return servicePrice;

}

}

sealed public class Waybill : Document, IDocument //накладная

{

private int servicePrice;

public Waybill(string title, DateTime date, Client client, Organization organization, int servicePrice, int id) : base(title, date, client, organization, id)

{

this.servicePrice = servicePrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + servicePrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + servicePrice);

}

override public int GetTotalPrice()

{

return servicePrice;

}

}

sealed public class Check : Document, IDocument

{

private int totalPrice;

public Check(string title, DateTime date, Client client, Organization organization, int totalPrice, int id) : base(title, date, client, organization, id)

{

this.totalPrice = totalPrice;

}

public override string ToString()

{

return Title + " " + Date.ToString("MM/dd/yyyy") + " " + Name + " " + NameOfOrganization + " " + totalPrice;

}

public override void Info()

{

Console.WriteLine(Title + "\n" + "Дата заключения: " + Date.ToString("MM/dd/yyyy") + "\n" + "Клиент: " + Name + "\n" + "Организация: " + NameOfOrganization + "\n" + "Итоговая стоимость: " + totalPrice);

}

override public int GetTotalPrice()

{

return totalPrice;

}

}

public class Client

{

private string name;

public Client(string name)

{

this.name = name;

}

public string Name

{

get { return name; }

set { name = value; }

}

}

class IsNotNameOfOrganization : ArgumentException

{

string Value { get; set; }

public IsNotNameOfOrganization(string message, string value) : base(message)

{

Value = value;

}

}

class IsNotTitle : ArgumentException

{

string Value { get; set; }

public IsNotTitle(string message, string value) : base(message)

{

Value = value;

}

}

class WrongIdValue : Exception

{

int Value { get; set; }

public WrongIdValue(string message, int value) : base(message)

{

Value = value;

}

}

public class Organization

{

private string nameOfOrganization;

public Organization(string nameOfOrganization)

{

if (nameOfOrganization.Length <= 1)

{

throw new IsNotNameOfOrganization("Недопустимое значение для имени организации", nameOfOrganization);

}

else this.nameOfOrganization = nameOfOrganization;

}

public string NameOfOrganization

{

get { return nameOfOrganization; }

set { nameOfOrganization = value; }

}

}

}

**LABA 8**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace laba8

{

class Program

{

interface IInterface<T>

{

CollectionType<T> add(CollectionType<T> mass1, CollectionType<T> mass2);

CollectionType<T> delete(CollectionType<T> mass2);

CollectionType<T> outmass(CollectionType<T> mass2);

}

static void Main(string[] args)

{

try

{

CollectionType<int> mass1 = new CollectionType<int>() { abc = { 1, 6, 4, 7, 9, 3 } };

CollectionType<int> mass2 = new CollectionType<int>() { abc = { 6, 1, 7, 5 } };

mass2.add(mass1, mass2);

string way = @"/Users/Alex/Univercity/OOP/laba8/laba8/text.txt";

using (StreamWriter file = new StreamWriter(way))

{

foreach (int i in mass2.abc)

{

file.WriteLine(i);

}

}

using (StreamReader str = new StreamReader(way))

{

string strall;

while ((strall = str.ReadLine()) != null)

{

Console.WriteLine(strall);

}

}

}

catch (Exception ex)

{

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

}

finally

{

Console.WriteLine("Finally!");

}

}

public class CollectionType<T> : IInterface<T>

{

public List<T> abc = new List<T> { };

public CollectionType<T> add(CollectionType<T> mass1, CollectionType<T> mass2)

{

if ((mass1.abc != null && mass1.abc.Count > 0) && (mass2.abc != null && mass2.abc.Count > 0))

{

mass2.abc.AddRange(mass1.abc);

}

mass2.abc = mass2.abc.ToList();

return mass2;

}

public CollectionType<T> delete(CollectionType<T> mass2)

{

if (mass2.abc != null && mass2.abc.Count > 0)

{

mass2.abc.RemoveAt(Count(mass2));

}

return mass2;

}

public CollectionType<T> outmass(CollectionType<T> mass2)

{

foreach (T i in mass2.abc)

{

Console.WriteLine(i);

}

return mass2;

}

public static int Count(CollectionType<T> mass2)

{

int counter = 0;

foreach (T i in mass2.abc)

{

counter++;

}

return counter;

}

public static CollectionType<T> operator +(CollectionType<T> mass1, CollectionType<T> mass2)

{

CollectionType<T> mass3 = new CollectionType<T>() { };

if ((mass1.abc != null && mass1.abc.Count > 0) && (mass2.abc != null && mass2.abc.Count > 0))

{

mass3.abc.AddRange(mass1.abc);

mass3.abc.AddRange(mass2.abc);

}

mass3.abc = mass3.abc.ToList();

return mass3;

}

public static explicit operator int(CollectionType<T> set)

{

int x = set.abc.Count();

return x;

}

public class CollectionA<T>

{

}

public class CollectionB<T> where T : CollectionA<int>

{

}

}

}

}

**LABA 9**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegate

{

public delegate void BeLate(string str);

public delegate void TakeMoney(object obj, EventArgs args);

public delegate void Promotion(object obj, EventArgs args, string str);

public delegate void Balance(Student b);

class Program

{

static void Main(string[] args)

{

BeLate massage = (str) => Console.WriteLine(str);

Balance balance = (show) => Console.WriteLine($"Текущая зарплата: { show.Salary} рублей");

Dekan dekan = new Dekan();

Student student1 = new Student(500, "студент");

Undergraduate undergraduate1 = new Undergraduate(600, "магистрант");

student1.Late();

student1.RegisterHandler(massage);

student1.Late();

balance(student1);

dekan.money += student1.Money;

dekan.money += undergraduate1.Money;

dekan.newMoney();

student1.Late();

student1.Late();

balance(student1);

dekan.newMoney();

balance(student1);

dekan.promotion += student1.ToPromoteStudent;

dekan.ToPromote("выпускник-студент");

dekan.promotion -= student1.ToPromoteStudent;

dekan.promotion += undergraduate1.ToPromoteUndergraduate;

dekan.ToPromote("выпускник-магистрант");

balance(undergraduate1);

Console.WriteLine();

string str1 = "dadOda dadOda";

Func<string, string> func;

func = ChangeString.Delet;

str1 = func(str1);

Console.WriteLine(str1);

Action<string, char> action;

action = ChangeString.AddLetter;

action(str1, 's');

func += ChangeString.Oa;

str1 = func(str1);

Console.WriteLine(str1);

func += ChangeString.SToApper;

str1 = func(str1);

Console.WriteLine(str1);

func += ChangeString.SToLower;

str1 = func(str1);

Console.WriteLine(str1);

}

}

}

//////////////////////////////////////////

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegate

{

public class Dekan //класс

{

public event TakeMoney money; //переменные класса

public event Promotion promotion;

public void newMoney() //метод

{

Console.WriteLine("Проверяем пропуски...");

money?.Invoke(this, null); //метод инвоук, принимающий делегат

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

}

public void ToPromote(string newPosotion) //метод

{

Console.WriteLine("Назначаем новую должность...");

promotion?.Invoke(this, null, newPosotion);

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

}

}

public class Student

{

private int salary;

private string position;

public int lateCounter = 0;

BeLate late;

public void RegisterHandler(BeLate late)

{

this.late = late;

}

public int Salary

{

get { return salary; }

set { salary = value; }

}

public string Position

{

get { return position; }

set { position = value; }

}

public Student() { }

public Student(int salary, string position)

{

Salary = salary;

Position = position;

}

public void Late()

{

lateCounter++;

Console.WriteLine("Студент опоздал!");

late?.Invoke($"Количество опозданий: {lateCounter}");

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

}

public virtual void Money(object sender, EventArgs e) //метод проверки кол-ва пропусков

{

if (lateCounter >=3)

{

Console.WriteLine("Слишком много опозданий, штраф 10 рублей");

Salary -= 10;

}

else Console.WriteLine("Допустимое количество опазданий");

}

public void ToPromoteStudent(object sender, EventArgs e, string str) //изменение должности

{

Position = str;

Console.WriteLine($"Новая должность: {Position}");

}

}

public class Undergraduate : Student //наследуемый класс

{

public Undergraduate(int salary, string position) : base(salary, position)

{

}

public override void Money(object sender, EventArgs e)

{

if (lateCounter >= 3)

{

Console.WriteLine("Слишком много опозданий, штраф 10 рублей");

Salary -= 10;

}

else Console.WriteLine("Допустимое количество опазданий");

}

public void ToPromoteUndergraduate(object sender, EventArgs e, string str)

{

Position = str;

Console.WriteLine($"Новая должность: {Position}");

}

}

static public class ChangeString

{

public static string SToApper(string str) //большие буквы

{

str = str.ToUpper();

return str;

}

public static string SToLower(string str) //маленькие буквы

{

str = str.ToLower();

return str;

}

public static string Delet(string str) //удаление символа

{

int del = 11;

str = str.Remove(del, 3);

return str;

}

public static void AddLetter(string str, char letter) //добавление символа

{

str = str + letter;

Console.WriteLine(str);

}

public static string Oa(string str) //изменение символа

{

str = str.Replace("O", "a");

return str;

}

}

}

**LABA 10**

using System;

using System.Collections;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace laba10

{

public class Game<T> : IEnumerable<T> where T : Player //наследуемый класс

{

public BlockingCollection<T> players = new BlockingCollection<T>();

public Dictionary<int, T> dict = new Dictionary<int, T>();

public Player winner;

Random rnd = new Random();

public void StartGame() //старт игры

{

foreach (var item in players)

{

item.number = rnd.Next(1, 1000);

}

winner = players.OrderByDescending(i => i.number).First(); //операция упорядочевания

}

public void Show() //показ игроков

{

foreach (var item in players)

{

Console.WriteLine(item);

}

}

IEnumerator IEnumerable.GetEnumerator() //получение счетчика

{

return this.GetEnumerator();

}

IEnumerator<T> GetEnumerator()

{

foreach (T foo in this.players)

{

yield return foo;

}

}

IEnumerator<T> IEnumerable<T>.GetEnumerator()

{

return ((IEnumerable<T>)players).GetEnumerator();

}

}

public class Player //класс игрок

{

public string name; //переменные класса

public int number;

public Player(string name)

{

this.name = name;

this.number = 0;

}

public override string ToString() //вывод числа игрока

{

return name + " создал число " + number;

}

}

//№2-6

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

namespace \_2\_6

{

class Program

{

static void Main(string[] args)

{

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

Users.Add(new User("Ivan", "a@example.com", "123"));

Users.Add(new User("Petr", "c@example.com", "123"));

Users.Add(new User("Gosha", "a@example.com", "123"));

Users.Add(new User("Daniil", "a@example.com", "123"));

Users.Add(new User("Alexey", "op@example.com", "123"));

for (int i = 0; i < Users.Count - 1; i++)

Console.WriteLine($"Результат сравнения {Users[i]} и {Users[i + 1]}: {Users[i].CompareTo(Users[i + 1])}");

foreach (var user in Users)

{

WebNet.Add(user);

Console.WriteLine(user.Status);

}

Console.WriteLine("Count: " + WebNet.CountOfSingIn());

WebNet.Serialization();

}

public static class WebNet

{

private static LinkedList<User> GitHub = new LinkedList<User>();

public static void Add(User u)

{

GitHub.AddLast(u);

}

public static void Remove(User u)

{

GitHub.Remove(u);

}

public static int CountOfSingIn()

{

return GitHub.Where(x => x.Status == Statuses.SingnIn).Count();

}

public static void Serialization()

{

using (FileStream fs = new FileStream("Users.bin", FileMode.Create))

{

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

foreach (var user in GitHub.Where(x => x.Status == Statuses.SingnIn))

Users.Add(user);

//XmlSerializer xml = new XmlSerializer(Users.GetType());

//xml.Serialize(fs, Users);

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(fs, Users);

}

}

}

[Serializable]

public class User : IComparable

{

public User(string name, string email, string password)

{

Name = name;

Email = email;

Password = password;

Random rnd = new Random();

Status = (Statuses)rnd.Next(0, 2);

ID = id++;

}

private static int id = 1;

public string Name { get; set; }

public int ID { get; private set; }

private string Email { get; set; }

private string Password { get; set; }

public Statuses Status { get; set; }

public int CompareTo(object obj)

{

return Email.CompareTo(((User)obj).Email);

}

public override bool Equals(object obj)

{

return ID == ((User)obj).ID;

}

public override int GetHashCode()

{

return ID;

}

public override string ToString()

{

return Name;

}

}

public enum Statuses

{

SingnIn,

SingOut

}

}

}

//end

class Program

{

public static void Ch(object sender, NotifyCollectionChangedEventArgs e)

{

Console.WriteLine("Коллекция изменилась с действием: " + e.Action);

}

static void Main(string[] args)

{

Player firstPlayer = new Player("Леша"); //создание игроков

Player secondPlayer = new Player("Артем");

Player thirdPlayer = new Player("Женя");

Player foursPlayer = new Player("Дима");

Game<Player> RollGame = new Game<Player>();

RollGame.players.Add(firstPlayer); //вызовы функций

RollGame.players.Add(secondPlayer);

RollGame.players.TryAdd(thirdPlayer);

RollGame.players.TryAdd(foursPlayer);

/\*RollGame.players.Take();\*/

RollGame.players.CompleteAdding();

RollGame.StartGame();

RollGame.Show();

Console.WriteLine("Игрок-победитель - " + RollGame.winner.name);

BlockingCollection<int> test = new BlockingCollection<int>();

test.Add(1);

test.Add(5);

test.TryAdd(6);

int x;

test.TryTake(out x);

foreach (var item in test)

Console.WriteLine(item);

RollGame.dict.Add(1, firstPlayer);

RollGame.dict.Add(2, secondPlayer);

RollGame.dict.Add(3, thirdPlayer);

RollGame.dict.Add(4, foursPlayer);

foreach (KeyValuePair<int, Player> element in RollGame.dict)

Console.WriteLine("Ключ: {0}. Значение: {1}", element.Key, element.Value);

if (RollGame.dict.ContainsValue(secondPlayer))

Console.WriteLine("Содержит значение");

else Console.WriteLine("Не содержит значение");

ObservableCollection<int> obs = new ObservableCollection<int>();

obs.CollectionChanged += Ch;

obs.Add(5);

obs.Remove(5);

}

}

}

**LABA 11**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LINQ

{

//№8-1

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

namespace \_8\_1

{

class Program

{

static void Main(string[] args)

{

Shop Evroopt = new Shop("Евроопт");

Evroopt.Add(new Item("Кока-кола 2л", 3.09f));

Evroopt.Add(new Item("Спрайт 1л", 1.79f));

Evroopt.Add(new Item("Фанта 0.5л", 1.19f));

Console.WriteLine("Магазин: " + Evroopt);

foreach (var item in Evroopt)

Console.WriteLine(item.Name + ":\t" + item.Price + " BYN");

var Enumerator = Evroopt.GetEnumerator();

Enumerator.MoveNext();

Manager.Sale += new Manager.SaleHandler(Enumerator.Current.OnSale);

Enumerator.MoveNext();

Manager.Sale += new Manager.SaleHandler(Enumerator.Current.OnSale);

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

Manager.CommandSale();

Console.WriteLine("Магазин: " + Evroopt);

foreach (var item in Evroopt)

Console.WriteLine(item.Name + ":\t" + item.Price + " BYN");

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

Console.WriteLine("Linq запрос: " + Evroopt.Where(x => x.Name == "Фанта 0.5л").Select(x => x.Price).First());

}

class Item

{

public Item(string name, float price)

{

ID = ++id;

Name = name;

Price = price;

}

private static int id = 0;

public string Name { get; set; }

public int ID { get; set; }

public float Price { get; set; }

public void OnSale()

{

Price = (float)Math.Round(Price / 2, 2);

}

}

class Shop : IEnumerable<Item>

{

public Shop(string name)

{

Name = name;

}

public string Name { get; }

private Queue<Item> Items = new Queue<Item>();

public void Add(Item item)

{

Items.Enqueue(item);

}

public Item Remove()

{

return Items.Dequeue();

}

public void Clear()

{

Items.Clear();

}

public IEnumerator<Item> GetEnumerator()

{

return Items.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return Items.GetEnumerator();

}

public override string ToString()

{

return Name;

}

public override bool Equals(object obj)

{

return ((Shop)obj).Name == Name;

}

public static Shop operator +(Shop shop, Item item)

{

shop.Add(item);

return shop;

}

public static Shop operator -(Shop shop, Item item)

{

shop.Remove();

return shop;

}

}

public static class Manager

{

public delegate void SaleHandler();

public static event SaleHandler Sale;

public static void CommandSale()

{

Sale();

}

}

}

}

//end

class Program

{

static void Main(string[] args)

{

string[] months = new string[] { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" }; //строки с месяцами

Console.WriteLine("Введите количество символов, которое должно совпадать с символами месяцев");

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

IEnumerable<string> Months1 = months.Where(m => m.Length == n);

foreach (string item in Months1)

{

Console.WriteLine(item);

}

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

Console.WriteLine("Только летние и зимние месяцы:");

IEnumerable<string> Months2 = from m in months where m.StartsWith("J") || m.StartsWith("F") || m.StartsWith("Au") || m.StartsWith("D") select m; //вывод месяцев по первым буквам

foreach (string item in Months2)

{

Console.WriteLine(item);

}

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

Console.WriteLine("Месяц в алфавитном порядке:");

IEnumerable<string> Months3 = months.OrderBy(s => s); //вывод по алфавиту

foreach (string item in Months3)

{

Console.WriteLine(item);

}

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

Console.WriteLine("Месяцы с \"u\" и длиной строки не менне 4ех: ");

IEnumerable<string> Months4 = months.Where(n1 => n1.Contains("u") && n1.Length >= 4); //4 и больше буквы

foreach (string item in Months4)

{

Console.WriteLine(item);

}

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

Book book1 = new Book("Лесное чудище", "Дима", "Беларусь", 2001, 241, 44, "тонкий", 3); //создание книг

Book book2 = new Book("Обитель Анубиса", "Саша", "Германия", 1995, 321, 32, "широкий", 6);

Book book3 = new Book("Важный человек", "Дима", "США", 1950, 321, 100, "тонкий", 4);

Book book4 = new Book("Величайший шоумен", "Женя", "Италия", 2010, 321, 28, "широкий", 2);

Book book5 = new Book("Ведьмы", "Алина", "Норвегия", 1991, 321, 74, "тонкий", 8);

List<Book> library = new List<Book>();

library.Add(book1); //добваление в библиотеку

library.Add(book2);

library.Add(book3);

library.Add(book4);

library.Add(book5);

Console.WriteLine("Введите автора для поиска:");

string findAuthor = Console.ReadLine();

var lib1 = library.Where(n3 => n3.Author == findAuthor); //поиск по автору

foreach (Book item in lib1)

{

Console.WriteLine(item.Author + " " + item.Title + " " + item.Year);

}

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

Console.WriteLine("Введите, после какого года необходим список книг:");

int year = int.Parse(Console.ReadLine()); //поиск по году

var lib2 = library.Where(n3 => n3.Year > year);

foreach (Book item in lib2)

{

Console.WriteLine(item.Author + " " + item.Title + " " + item.Year);

}

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

Console.WriteLine("Cписок книг отсортированных по цене..."); //сортировка по цене

var lib3 = library.OrderBy(n4 => n4.Cost);

foreach (Book item in lib3)

{

Console.WriteLine(item.Author + " " + item.Title + " " + item.Cost);

}

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

var lib4 = library.Min(n5 => n5.BlindingTypeNumber);

Console.WriteLine("Самая тонкая книга имеет толщину: " + lib4); //толщина книги

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

Console.WriteLine("5 первых самых толстых книг...");

var lib5 = library.OrderByDescending(n5 => n5.BlindingTypeNumber).Take(5);

foreach (Book item in lib5)

{

Console.WriteLine(item.Author + " " + item.Title + " " + "толщина: " + item.BlindingTypeNumber);

}

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

List<Team> teams = new List<Team>()

{

new Team { Name = "Оттава Сенаторз", Country ="Канада" }, //создание команды

new Team { Name = "Сиэтл Кракен", Country ="США" }

};

List<Player> players = new List<Player>()

{

new Player {Name="Мэтт Мюррей", Team="Оттава Сенаторз"}, //создание игроков

new Player {Name="Тома Шабо", Team="Оттава Сенаторз"},

new Player {Name="Томми Томпсон", Team="Сиэтл Кракен"}

};

var result = players.Join(teams, p => p.Team, t => t.Name, (p, t) => new { Name = p.Name, Team = p.Team, Country = t.Country });

foreach (var item in result)

Console.WriteLine($"{item.Name} - {item.Team} ({item.Country})");

}

}

}

////////////////////////////////////////////////

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LINQ

{

public class BelBook //класс белкнига

{

public string BelAuthor { get; set; }

public string BelTitle { get; set; }

public BelBook(string belAuthor, string belTitle)

{

BelAuthor = belAuthor;

BelTitle = belTitle;

}

}

class Player //класс игрок

{

public string Name { get; set; }

public string Team { get; set; }

}

class Team //класс команда

{

public string Name { get; set; }

public string Country { get; set; }

}

public class Book //класс книга

{

public string title; //переменные книги

public string author;

private string publishingHouse;

public ushort year;

private uint pageNumber;

private uint cost;

private string blindingType;

private int blindingTypeNumber;

public Book(string title, string author, string publishingHouse, ushort year, uint pageNumber, uint cost, string blindingType, int blindingTypeNumber)

{

this.title = title; //задание значений свойствам

this.author = author;

this.publishingHouse = publishingHouse;

this.year = year;

this.pageNumber = pageNumber;

this.cost = cost;

this.blindingType = blindingType;

this.blindingTypeNumber = blindingTypeNumber;

}

public string Title //название

{

get { return title; }

set { title = value; }

}

public string Author //автор

{

get { return author; }

set { author = value; }

}

public string PublishingHouse //страна

{

get { return publishingHouse; }

set { publishingHouse = value; }

}

public ushort Year //год

{

get { return year; }

set

{

if (year <= 2020)

year = value;

else year = 0;

}

}

public uint PageNumber //кол-во страниц

{

get { return pageNumber; }

set { pageNumber = value; }

}

public uint Cost //цена

{

get { return cost; }

set { cost = value; }

}

public string BlindingType //тип переплета

{

get { return blindingType; }

set { blindingType = value; }

}

public int BlindingTypeNumber //номер типа

{

get { return blindingTypeNumber; }

set { blindingTypeNumber = value; }

}

}

}

**LABA 12**

using System;

using System.Collections.Generic;

using System.Dynamic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace laba12

{

public interface Show //интерфейс

{

void Show();

}

public class Test : Show

{

public string name;

public Test(string name) //строка имени

{

this.name = name;

}

public Test()

{

this.name = null;

}

public void Show() //вывод имени

{

Console.WriteLine(name);

}

public void ToConsole(List<string> spisok) //вывод в консоль

{

foreach (string str in spisok)

{

Console.WriteLine(str);

}

}

public override string ToString()

{

return "Объект класса Test с именем " + name;

}

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

}

static class Reflector //класс рефлексии

{

static public void ClName(string classname) //класс вывода сборки

{

Assembly tekushchayaAssembly = Assembly.GetExecutingAssembly();

Type t = tekushchayaAssembly.GetType(classname);

Assembly assem = t.Assembly;

Console.WriteLine("Полное имя сборки: ");

Console.WriteLine(assem.FullName);

Console.WriteLine();

Console.WriteLine("Расположение сборки: ");

Console.WriteLine(assem.CodeBase);

}

static public void GetConstructor(string classname) //класс показывающий конструкторы

{

ConstructorInfo[] p = Type.GetType(classname).GetConstructors();

Console.WriteLine();

Console.WriteLine("Имеет ли класс конструкторы?");

if (p.Length > 0)

Console.WriteLine($"Да, {p.Length}");

else

Console.WriteLine("Нет!");

}

static public void Publi(string classname) //класс показывающий методы

{

Type t = Type.GetType(classname);

Console.WriteLine();

Console.WriteLine("Список методов: ");

foreach (MethodInfo cMethod in t.GetMethods())

{

Console.WriteLine(cMethod.Name);

}

}

static public void Field(string classname) //класс показывающий поля

{

Type t = Type.GetType(classname);

Console.WriteLine();

Console.WriteLine("Список полей: ");

foreach (FieldInfo fInfo in t.GetFields())

{

Console.WriteLine(fInfo.FieldType.Name + " " + fInfo.Name);

}

Console.WriteLine();

Console.WriteLine("Список свойств: ");

foreach (PropertyInfo pInfo in t.GetProperties())

{

Console.WriteLine(pInfo.PropertyType.Name + " " + pInfo.Name);

}

}

static public void Interface(string classname) //класс показывающий интерфейсы

{

Type t = Type.GetType(classname);

Console.WriteLine();

Console.WriteLine("Список интерфейсов: ");

foreach (Type tp in t.GetInterfaces())

{

Console.WriteLine(tp.Name);

}

}

static public void MethodForType(string classname, string parametr) //класс показывающий методы с определенным типом

{

Type t = Type.GetType(classname);

MethodInfo[] methods = t.GetMethods();

Console.WriteLine();

Console.WriteLine("Методы класса {0} с типом аргумента: {1}", classname, parametr);

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

{

ParameterInfo[] param = methods[i].GetParameters();

for (int j = 0; j < param.Length; j++)

{

if (parametr == param[j].ParameterType.Name)

{

Console.WriteLine(methods[i].Name);

}

}

}

}

public static void CallMethod(string className, string methodName) //класс выполнения метода

{

Type type = Type.GetType(className);

List<string> FirstParam = File.ReadAllLines(@"/Users/Alex/Univercity/OOP/laba12/laba12/Refl.txt").ToList();

List<string>[] parametrs = new List<string>[] { FirstParam };

try

{

object obj = Activator.CreateInstance(type);

MethodInfo method = type.GetMethod(methodName);

Console.WriteLine();

Console.WriteLine("Результат выполнения метода: ");

Console.WriteLine(method.Invoke(obj, parametrs));

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

static public void Create(string classname, string name) //класс создания

{

Type t = Type.GetType(classname);

ConstructorInfo[] p = Type.GetType(classname).GetConstructors();

object obj = Activator.CreateInstance(t, args: name);

Console.WriteLine(obj.ToString());

}

}

//№ 3-2

using System;

using System.Collections.Generic;

using System.Linq;

using System.IO;

using System.Xml.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace \_3\_2

{

class Program

{

static void Main(string[] args)

{

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

Uber.Add(new Taxi() { Number = 1, Position = new Location() { Lat = 12.3439493f, Long = 483.129849384f, Speed = 63.2f }, Status = Statuses.busy });

Uber.Add(new Taxi() { Number = 2, Position = new Location() { Lat = 42.3439493f, Long = 452.129849384f, Speed = 0f }, Status = Statuses.free });

Uber.Add(new Taxi() { Number = 3, Position = new Location() { Lat = 22.3439493f, Long = 495.129849384f, Speed = 44.2f }, Status = Statuses.free });

Uber.Add(new Taxi() { Number = 4, Position = new Location() { Lat = 17.3439493f, Long = 457.129849384f, Speed = 68f }, Status = Statuses.busy });

Console.WriteLine("---Введите свои координаты---");

Location UserLocation = new Location();

Console.Write("Широта: ");

UserLocation.Lat = float.Parse(Console.ReadLine());

Console.Write("Долгота: ");

UserLocation.Long = float.Parse(Console.ReadLine());

List<Taxi> taxis = new List<Taxi>(Uber.park);

var TaxisToUser = taxis.OrderBy(x => Math.Sqrt(Math.Pow(x.Position.Lat - UserLocation.Lat, 2) + Math.Pow(x.Position.Long - UserLocation.Long, 2)));

foreach (var taxi in TaxisToUser)

Console.WriteLine(taxi.Number);

using (FileStream sw = new FileStream("Taxi.bin", FileMode.Create))

{

//XmlSerializer xml = new XmlSerializer(TaxisToUser.First().GetType());

//xml.Serialize(sw, TaxisToUser.First());

BinaryFormatter bin = new BinaryFormatter();

bin.Serialize(sw, TaxisToUser.First());

}

}

[Serializable]

public class Location

{

public float Lat { get; set; }

public float Long { get; set; }

public float Speed { get; set; }

}

[Serializable]

public class Taxi

{

public int Number { get; set; }

public Location Position { get; set; }

public Statuses Status { get; set; }

}

[Serializable]

public enum Statuses

{

busy,

free

}

public class Park<T>

{

public List<T> park { get; } = new List<T>();

public void Add(T obj)

{

park.Add(obj);

}

public void Remove(int index)

{

park.RemoveAt(index);

}

public void Clear()

{

park.Clear();

}

public T Find(Predicate<T> predicate)

{

foreach (var obj in park)

{

if (predicate(obj))

return obj;

}

return default(T);

}

}

}

}

//end

class Program

{

static void Main(string[] args)

{

Reflector.ClName("laba12.Test"); //вызовы функций

Reflector.GetConstructor("laba12.Test");

Reflector.Publi("laba12.Test");

Reflector.Field("laba12.Test");

Reflector.Interface("laba12.Test");

Reflector.MethodForType("laba12.Test", "Int32");

Reflector.CallMethod("laba12.Test", "ToConsole");

Reflector.Create("laba12.Test", "Alex FIT");

}

}

}

**LABA 13**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab13

{

class Program

{

static void Main(string[] args)

{

TASDiskInfo diskInfo = new TASDiskInfo();

diskInfo.DiskInfo();

TASFileInfo fileInfo = new TASFileInfo();

fileInfo.FileData(@"/Users/Alex/Univercity/OOP/laba13/laba13/Class1.cs");

TASDirInfo dirInfo = new TASDirInfo();

dirInfo.DirInfo(@"/Users/Alex/Univercity");

TASFileManager fileManager = new TASFileManager();

fileManager.FileManager("/Users");

TASLog.SearchByString("FileInfo:");

}

}

}

//////////////////////////////////////

using System;

using System.Collections.Generic;

using System.IO;

using System.IO.Compression;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab13

{

public static class TASLog

{

public const string sourceFile = @"/Users/Alex/Univercity/OOP/laba13/laba13/aaaa.txt"; //путь к файлу

static TASLog()

{

using (StreamWriter w = new StreamWriter(sourceFile, false)) { }

}

public static void WriteLine(string str)

{

try //отлов ошибки

{

using (StreamWriter w = new StreamWriter(sourceFile, true)) //создание потока

{

w.WriteLine(str);

}

}

catch (Exception ex)

{

Console.WriteLine("Exception: " + ex.Message);

}

}

public static void SearchByString(string str) //поиск по строке

{

using (StreamReader sr = new StreamReader(sourceFile, false)) //читатель строк

{

while (!sr.EndOfStream)

{

if (sr.ReadLine().StartsWith(str))

Console.WriteLine(sr.ReadLine());

}

}

}

}

public class TASDiskInfo

{

public void DiskInfo()

{

TASLog.WriteLine("Информация о диске:");

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

TASLog.WriteLine("\tИмя: " + drive.Name); //вывод инфомации о диске

TASLog.WriteLine("\tТип: " + drive.DriveType);

if (drive.IsReady)

{

TASLog.WriteLine("\tФайловая система: " + drive.DriveFormat);

TASLog.WriteLine($"\tОбъем свободного места: всего - {drive.TotalFreeSpace / 1000 / 1000 / 1000} GB, доступно - { drive.AvailableFreeSpace / 1024 / 1024 / 1024} GB");

TASLog.WriteLine($"\tОбщий размер: {drive.TotalSize / 1024 / 1024 / 1024} GB");

TASLog.WriteLine("\tМетка тома диска: " + drive.VolumeLabel);

}

TASLog.WriteLine("");

}

}

}

public class TASFileInfo

{

public void FileData(string path)

{

TASLog.WriteLine("Информация о файле:"); //метод вывода инфомации о файле

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

TASLog.WriteLine($"\tПолный путь: {fileInf.DirectoryName}");

TASLog.WriteLine($"\tИмя: {fileInf.Name}");

TASLog.WriteLine($"\tОбъем: {fileInf.Length}\n\tРасширение: {fileInf.Extension}\n\tДата создания: {fileInf.CreationTime}");

}

else

{

TASLog.WriteLine("Такого файла не существует");

}

}

}

public class TASDirInfo

{

public void DirInfo(string dirName)

{

DirectoryInfo dirInfo = new DirectoryInfo(dirName); //вывод инфомации о директории

TASLog.WriteLine("\nИнформация о директории:");

TASLog.WriteLine($"\tКоличество файлов: {dirInfo.GetFiles().Count()}");

TASLog.WriteLine($"\tДата создания: {dirInfo.CreationTime}");

TASLog.WriteLine($"\tПодкаталоги: {dirInfo.GetDirectories("\*", SearchOption.AllDirectories).Count()}");

TASLog.WriteLine($"\tParents: {dirInfo.Parent}");

}

}

public class TASFileManager

{

public void FileManager(string path) //метод вывода инфомации о директориях и файлах

{

try

{

DriveInfo driveInfo = new DriveInfo(path);

DirectoryInfo dirInfo = Directory.CreateDirectory(driveInfo.Name + "ALIInspect");

using (StreamWriter writer = File.CreateText(dirInfo.FullName + "\\alidirinfo.txt"))

{

writer.WriteLine("Информация");

}

File.Copy(dirInfo.FullName + "\\alidirinfo.txt", dirInfo.FullName + "\\copied.txt");

File.Delete(dirInfo.FullName + "\\alidirinfo.txt");

DirectoryInfo dirInfo1 = Directory.CreateDirectory(driveInfo.Name + "ALIFiles");

DirectoryInfo currentDirectory = new DirectoryInfo("./");

foreach (var item in currentDirectory.GetFiles())

item.CopyTo(dirInfo1.FullName + "\\" + item.Name, true);

dirInfo1.MoveTo(dirInfo.FullName + "\\" + dirInfo1.Name);

DirectoryInfo dirInfo2 = new DirectoryInfo(dirInfo.FullName + "\\" + dirInfo1.Name);

ZipFile.CreateFromDirectory(dirInfo2.FullName, dirInfo.FullName + "\\arhive.zip");

DirectoryInfo exInfo = Directory.CreateDirectory(dirInfo.FullName + "\\Ezvlechen");

using (ZipArchive arch = ZipFile.OpenRead(dirInfo.FullName + "\\arhive.zip"))

{

foreach (ZipArchiveEntry entry in arch.Entries)

{

entry.ExtractToFile(exInfo.FullName + "\\Ezvlechen\_" + entry.Name);

}

}

}

catch (Exception ex)

{

Console.WriteLine("Exception: " + ex.Message);

}

}

}

}

**LABA 14**

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Xml.Serialization;

using System.Runtime.Serialization.Json;

using System.Xml;

using System.Xml.Linq;

using System.Collections.Generic;

using System.Linq;

//Сериализация позволяет разработчику сохранять состояние объекта и воссоздавать его при необходимости.

//Это полезно для длительного хранения объектов или для обмена данными.

namespace laba14

{

class Program

{

static void Main(string[] args)

{

Point point = new Point(5, 9);

Point point\_2 = new Point(7, 23);

Point[] points = { point, point\_2 };

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

BinaryFormatter formatter = new BinaryFormatter();//Сериализует и десериализует объект или полный граф связанных объектов в двоичном формате

using (FileStream fs = new FileStream("point.dat", FileMode.OpenOrCreate))//FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

formatter.Serialize(fs, point);//Сериализация — это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.

Console.WriteLine("Объект сериализован");

formatter.Serialize(fs, points);//Сериализация — это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл.

Console.WriteLine("Объект сериализован");

}

using (FileStream fs = new FileStream("point.dat", FileMode.OpenOrCreate))//FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

Point newPoint = (Point)formatter.Deserialize(fs);//получить из потока байтов ранее сохраненный объект

Console.WriteLine("Объект десериализован");

Console.WriteLine($"X: {newPoint.X}, Y: {newPoint.Y}");

Point[] newPoint2 = (Point[])formatter.Deserialize(fs);//получить из потока байтов ранее сохраненный объект

Console.WriteLine("Объект десериализован");

foreach (Point p in newPoint2)

{

Console.WriteLine($"X: {p.X}, Y: {p.Y}");

}

}

Console.WriteLine();

Console.WriteLine("\_\_\_\_\_\_\_\_\_Сериализация в XML\_\_\_\_\_\_\_\_\_");

XmlSerializer serializer = new XmlSerializer(typeof(Point));//Сериализует и десериализует объекты в XML-документы и из них. XmlSerializer позволяет контролировать способ кодирования объектов в XML

XmlSerializer serializer1 = new XmlSerializer(typeof(Point[]));//Сериализует и десериализует объекты в XML-документы и из них. XmlSerializer позволяет контролировать способ кодирования объектов в XML

using (FileStream fs = new FileStream("point2.xml", FileMode.OpenOrCreate))////FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

serializer.Serialize(fs, point);

Console.WriteLine("Объект сериализован");

}

using (FileStream fs = new FileStream("point2.xml", FileMode.OpenOrCreate))////FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

Point newPoint = (Point)serializer.Deserialize(fs);

Console.WriteLine("Объект десериализован");

Console.WriteLine($"X: {newPoint.X}, Y: {newPoint.Y}");

}

using (FileStream fs = new FileStream("point3.xml", FileMode.OpenOrCreate))////FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

serializer1.Serialize(fs, points);

Console.WriteLine("Объект сериализован");

}

using (FileStream fs = new FileStream("point3.xml", FileMode.OpenOrCreate))////FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

Point[] newPoint2 = (Point[])serializer1.Deserialize(fs);

Console.WriteLine("Объект десериализован");

foreach (Point p in newPoint2)

{

Console.WriteLine($"X: {p.X}, Y: {p.Y}");

}

}

Console.WriteLine();

Console.WriteLine("\_\_\_\_\_\_\_\_\_Сериализация в JSON\_\_\_\_\_\_\_\_\_");

Person person1 = new Person("Tom", 29);

Person person2 = new Person("Will", 25);

Person[] people = { person1, person2 };

DataContractJsonSerializer jsonFormatter = new DataContractJsonSerializer(typeof(Person[]));//Сериализует объекты в нотацию объектов JavaScript (JSON) и десериализует данные JSON в объекты. Этот класс не наследуется

using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))//FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

jsonFormatter.WriteObject(fs, people);//Записывает все данные объекта (начальный XML-элемент, содержимое и закрывающий элемент) в XML-документ или поток

}

using (FileStream fs = new FileStream("people.json", FileMode.OpenOrCreate))//FileStream - синхронные и асинхронные операции чтения и записи. FileMode.OpenOrCreate - Указывает, что операционная система должна открыть файл, если он существует, в противном случае должен быть создан новый файл

{

Person[] newpeople = (Person[])jsonFormatter.ReadObject(fs);//Считывает XML-поток и возвращает десериализованный объект

foreach (Person p in newpeople)

{

Console.WriteLine("Имя: {0} --- Возраст: {1}", p.Name, p.Age);

}

}

Console.WriteLine();

Console.WriteLine("\_\_\_\_\_\_\_\_\_Linq to XML\_\_\_\_\_\_\_\_\_");

//создание документа

XDocument xdoc = new XDocument(new XElement("users",//создается документ, а затем в него добавляется комментарий и элемент

new XElement("user",

new XAttribute("name", "Bill Gates"),

new XElement("company", "Microsoft"),

new XElement("age", "48")),

new XElement("user",

new XAttribute("name", "Larry Page"),

new XElement("company", "Google"),

new XElement("price", "42"))));

xdoc.Save("users.xml");

IEnumerable<XElement> elements = xdoc.Descendants("user"); //Предоставляет перечислитель, который поддерживает простой перебор элементов неуниверсальной коллекции(kollectiya)//получаем отдельный эл-т

foreach (XElement e in elements)

Console.WriteLine("Элемент {0} : значение = {1}", e.Name, e.Value);

IEnumerable<XElement> elements2 = xdoc.Descendants("user").Where(e => ((string)e.Element("company")) == "Microsoft");

foreach (XElement e in elements2)

Console.WriteLine("Элемент {0} : значение = {1}", e.Name, e.Value);

Console.WriteLine();

//Используя XPath напишите два селектора для вашего XML документа

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

XmlDocument xml = new XmlDocument();

xml.Load("users.xml");

XmlElement xRoot = xml.DocumentElement;//только компании, все узлы корневого элемента//Возвращает корень XmlElement для документа

XmlNodeList list = xRoot.SelectNodes("//user/company");//Выбирает список узлов в соответствии с выражением XPath

foreach (XmlNode nodes in list)

{

Console.WriteLine(nodes.InnerText);//Возвращает или задает связанные значения узла и всех его дочерних узлов

}

XmlNode childnode = xRoot.SelectSingleNode("user[company='Microsoft']");//узел, у которого вложенный элемент "company" имеет значение "Microsoft"

if (childnode != null)

Console.WriteLine(childnode.OuterXml);

}

}

}

//////////////////////////////////

using System;

namespace laba14

{

[Serializable]

public class Point

{

public int X { get; set; }

public int Y { get; set; }

public Point()

{ }

public Point(int x, int y)

{

X = x;

Y = y;

}

}

}

//////////////////////////////////

using System;

using System.Runtime.Serialization;

namespace laba14

{

[DataContract]

public class Person

{

[DataMember]

public string Name { get; set; }

[DataMember]

public int Age { get; set; }

public Person(string name, int year)

{

Name = name;

Age = year;

}

}

}

**LABA 15**

using System;

using System.Diagnostics;

using System.IO;

using System.Reflection;

using System.Threading;

namespace laba15

{

class Program

{

public static bool Prostie(int n)

{

bool result = true;

if (n > 1)

{

for (uint i = 2u; i < n; i++)

{

if (n % i == 0)

{

result = false;

break;

}

}

}

else

{

result = false;

}

return result;

}

public static void Operation() //метод операции

{

int n = 100;

for (int i = 2; i <= n; i++)

{

if (Prostie(i))

{

Console.Write("Второй поток:");

Console.WriteLine(i);

}

}

}

public static void Event() //метод события

{

for (int i = 2; i < 100; i += 2)

{

Thread.Sleep(100); //время сна потока

Console.WriteLine(i);

}

Thread.Sleep(20);

}

public static void NEvent() //метод нет события

{

for (int i = 1; i < 100; i += 2)

{

Thread.Sleep(100);

Console.WriteLine(i);

}

Thread.Sleep(20);

}

public static void Count(object obj) //метод количество

{

int x = (int)obj;

for (int i = 1; i < 9; i++, x++)

{

Console.WriteLine($"{x \* i}");

}

}

static Mutex mutexObj = new Mutex();

static void Main(string[] args)

{

using (StreamWriter sw = new StreamWriter("prosesses.txt", false, System.Text.Encoding.Default))

{

foreach (Process p in Process.GetProcesses())

{

sw.WriteLine("Id " + p.Id); //вывод инфомации о потоках

sw.WriteLine("Name " + p.ProcessName);

sw.WriteLine("Priority " + p.BasePriority);

sw.WriteLine("Responding " + p.Responding);

sw.WriteLine("HandleConut " + p.HandleCount);

sw.WriteLine();

}

}

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

{

AppDomain app = AppDomain.CurrentDomain;

sw.WriteLine("Id: " + app.Id);

sw.WriteLine("Name: " + app.FriendlyName);

sw.WriteLine("Directory: " + app.BaseDirectory);

Assembly[] assApp = app.GetAssemblies();

foreach (Assembly item in assApp)

{

sw.WriteLine("Assembly name: " + item.FullName);

}

}

int n = 100;

Thread th = new Thread(new ThreadStart(Operation));

Console.WriteLine(th.ThreadState);

Console.WriteLine(th.Priority);

th.Name = "Second Thread";

Console.WriteLine(th.Name);

Console.WriteLine("id " + th.ManagedThreadId);

th.Start();

for (int i = 2; i <= n; i++)

{

if (Prostie(i))

{

Console.Write("Главный поток:");

Console.WriteLine(i);

}

}

Thread.Sleep(1000); //сон 1000 мс

Thread th1 = new Thread(new ThreadStart(Event));

Thread th2 = new Thread(new ThreadStart(NEvent));

th1.Priority = ThreadPriority.AboveNormal;

th1.Start();

th2.Start();

Console.ReadLine(); //чтение строки

int num = 0;

TimerCallback tm = new TimerCallback(Count);

Timer timer = new Timer(tm, num, 0, 2000);

Console.ReadLine();

}

}

}

**LABA 16**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections.Concurrent;

using System.Threading;

using System.Diagnostics;

using System.Threading.Tasks;

namespace laba16

{

class Program

{

static BlockingCollection<int> bc;

static async void SayAsync()//async - метод, который вызывает асинхронную функцию

{

await Task.Run(() => Console.WriteLine("Async stream"));//Оператор await приостанавливает вычисление до завершения асинхронной операции. После завершения асинхронной операции оператор await возвращает результат операции

}//Run Запускает стандартный цикл обработки сообщений приложения в текущем потоке

static void producer()

{

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

{

bc.Add(i \* i);//позволяет добавить новый элемент в конец списка

Console.WriteLine("Производится число " + i \* i);

}

bc.CompleteAdding();//Помечает экземпляры BlockingCollection<T> как не допускающие добавления дополнительных элементов

}

static void consumer()

{

int i;

while (!bc.IsCompleted)//Получает значение, указывающее, завершена ли задача

{

if (bc.TryTake(out i))//Пытается удалить элемент из коллекции BlockingCollection<T>

Console.WriteLine("Потребляется число: " + i);

}

}

public static Task task1;

public static void EasyNumbersIrato()//простые числа

{

Console.WriteLine("Current task ID: " + Task.CurrentId.ToString());//Возвращает идентификатор выполняющейся в настоящее время задачи Task

Console.WriteLine("Task Completed: " + task1.IsCompleted.ToString());//Получает значение, указывающее, завершена ли задача

Console.WriteLine("Status: " + task1.Status.ToString());//Получает состояние данной задачи

int i, j, n = 100;

int[] mas = new int[n];

for (i = 0; i < n; i++)

mas[i] = i + 1;

for (i = 1; i < n - 1; i++)

if (mas[i] != -1)

for (j = i + 1; j < n; j++)

if ((mas[j] != -1) && (mas[j] % mas[i] == 0))

mas[j] = -1;

for (i = 0; i < n; i++)

if (mas[i] != -1)

Console.WriteLine(mas[i]);

}

static int Factorial(int x)

{

int result = 1;

for (int i = 1; i <= x; i++)

{

result \*= i;

}

Console.WriteLine("Current task ID: " + Task.CurrentId.ToString());////Возвращает идентификатор выполняющейся в настоящее время задачи Task

Console.WriteLine("Result: " + result.ToString());

return result;

}

static int Sum(int a, int b, int c)

{

return a + b + c;

}

static void Main(string[] args)

{

Console.WriteLine("Задание 1");

Action action1 = new Action(EasyNumbersIrato);//delegate Инкапсулирует метод, который не имеет параметров и не возвращает значений

Stopwatch watch = Stopwatch.StartNew();//Предоставляет набор методов и свойств, которые можно использовать для точного измерения затраченного времени. StartNew Создает и запускает задачу

task1 = Task.Factory.StartNew(action1); //StartNew Создает и запускает задачу

task1.Wait();//Ожидает завершения выполнения задачи

task1.Dispose();//Реализация метода Dispose в основном используется для освобождения неуправляемых ресурсов.

watch.Stop();//вызов Stop завершает текущую меру интервала и замораживает совокупное значение затраченного времени

Console.WriteLine("Time for task: " + watch.Elapsed.ToString());//событие, которое Происходит по истечении интервала времени

Console.WriteLine("Task Completed: " + task1.IsCompleted.ToString());//Получает значение, указывающее, завершена ли задача

Console.WriteLine("Status: " + task1.Status.ToString());//Получает состояние данной задачи

Console.WriteLine();

Console.WriteLine("Задание 2");

CancellationTokenSource cancellation = new CancellationTokenSource();//создаем токен-ресурс, то есть отмену выполнения задачи

task1 = Task.Factory.StartNew(action1, cancellation.Token);////StartNew Создает и запускает задачу и используем токен-ресурс

cancellation.Cancel();//Возвращает или задает значение, показывающее, следует ли отменить событие

Console.WriteLine("Task Canceled: " + task1.IsCanceled.ToString());//Возвращает значение, указывающее, завершилось ли выполнение данного экземпляра Task из-за отмены

Console.WriteLine("Status: " + task1.Status.ToString());//Получает состояние данной задачи

Console.WriteLine();

Console.WriteLine("Задание 3 - 4");

Task<int> fact1 = new Task<int>(() => Factorial(5));//Task представляет собой работу над потоками для выполнения асинхронных операций

Task<int> fact2 = fact1.ContinueWith(a => Factorial(10));//Создает продолжение, которое выполняется асинхронно после завершения выполнения целевой задачи

Task<int> fact3 = fact2.ContinueWith(a => Factorial(15));

Task<int> res = new Task<int>(() => Sum(fact1.Result, fact2.Result, fact3.Result));

fact1.Start();//Запускает ресурс процесса и связывает его с компонентом Process

res.Start();//Запускает ресурс процесса и связывает его с компонентом Process

var awaiter = res.GetAwaiter();//Получает объект типа awaiter, используемый для данного объекта Task

awaiter.OnCompleted(() => Console.WriteLine("Last Result:" + awaiter.GetResult()));//Уведомляет наблюдателя о том, что поставщик завершил отправку push-уведомлений. GetResult Завершает ожидание завершения асинхронной задачи

Thread.Sleep(1000);

Console.WriteLine();

Console.WriteLine("Задание 5");

watch = Stopwatch.StartNew();//StartNew Создает и запускает задачу

int[] array = new int[100000000];

ParallelLoopResult result = Parallel.For(0, 10000000, (int z, ParallelLoopState loop) => { array[z] = z + 1; if (z == 1000) loop.Break(); });

//ParallelLoopResult Предоставляет состояние выполнения цикла Parallel. Parallel.For - Выполняет цикл for, в котором итерации могут выполняться параллельно

if (result.IsCompleted) Console.WriteLine("Выполнен");////Получает значение, указывающее, завершена ли задача

else Console.WriteLine("Выполнение цикла завершено на итерации {0}", result.LowestBreakIteration.ToString());//Получает первую итерацию цикла, из которой был прервал метод

watch.Stop();//вызов Stop завершает текущую меру интервала и замораживает совокупное значение затраченного времени

Console.WriteLine("Time for task: " + watch.Elapsed.ToString());////событие, которое Происходит по истечении интервала времени

Console.WriteLine();

Console.WriteLine("Задание 6");

Parallel.Invoke(() =>//Выполняет все предоставленные действия, по возможности в параллельном режиме

{

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

{

Console.Write("1 ");

}

},

() =>

{

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

{

Console.Write("2 ");

}

});

Console.WriteLine();

Console.WriteLine("\nЗадание 7");

bc = new BlockingCollection<int>(5);// Создадим задачи поставщика и потребителя

Task Pr = new Task(producer);

Task Cn = new Task(consumer);

Pr.Start();//Запускает ресурс процесса и связывает его с компонентом Process

Cn.Start();//Запускает ресурс процесса и связывает его с компонентом Process

try

{

Task.WaitAll(Cn, Pr);//Ожидает завершения выполнения всех указанных объектов Task

}

catch (Exception ex)

{

Console.WriteLine(ex);

}

finally

{

Cn.Dispose();////Реализация метода Dispose в основном используется для освобождения неуправляемых ресурсов

Pr.Dispose();

bc.Dispose();

}

Console.WriteLine("\nЗадание 8");

SayAsync();

Console.WriteLine("I\'m first!");

Console.ReadKey();

}

}

}