# № 2 Основы CLR и .NET. Типы. Массивы, кортежи и строки

using Microsoft.VisualBasic.FileIO;

using System;

using System.Runtime.CompilerServices;

using System.Text;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

// ТИПЫ

// Инициализация переменных

bool var1 = true;

byte var2 = 255;

sbyte var3 = -128;

short var4 = -2000;

ushort var5 = 102;

int var6 = 2147483643;

uint var7 = 0xFF; // 255

long var8 = 0b101; // 5

ulong var9 = 10;

float var10 = 55.3f;

double var11 = 193842397423749.91824;

decimal var12 = -1.2334234m;

char var13 = 'H';

string var14 = "Hello World!";

// Ввод строковой переменной

Console.Write("Введите переменую: ");

var14 = Console.ReadLine();

Console.WriteLine();

// Вывод переменных в консоль

Console.WriteLine($"bool: \t\t{var1}\n" +

$"byte: \t\t{var2}\n" +

$"sbyte: \t\t{var3}\n" +

$"short: \t\t{var4}\n" +

$"ushort: \t{var5}\n" +

$"int: \t\t{var6}\n" +

$"uint: \t\t{var7}\n" +

$"long: \t\t{var8}\n" +

$"ulong: \t\t{var9}\n" +

$"float: \t\t{var10}\n" +

$"double: \t{var11}\n" +

$"decimal: \t{var12}\n" +

$"char: \t\t{var13}\n" +

$"string: \t{var14}\n");

// Явные преобразования

var6 = (int)var10;

var11 = Convert.ToDouble(var2);

//var1 = Convert.ToBoolean(var13);

var14 = Convert.ToString(var13);

// Неявные преобразования

double d = 11.2;

int i = 92;

float f = 45.2f;

byte b = 72;

short s = 29;

long l = 12;

f = i;

i = b;

d = f;

i = s;

l = i;

// Упаковка и распаковка

i = 34;

object o = i;

int j = (int)o;

// Неявно типизированные переменные

var v = 5; // int

var str = "Hello World!"; // string

var a = new[] { 1, 2, 3 }; // array of int

// Nullable

int? x = null;

x = 14;

// Ошибка с var

var h = 5;

//h = 5.5f; // Невозможно так как предыдущая строка определила переменную h, как переменную типа int

// СТРОКИ

// Сравнение строк

string stringHello = "Hello",

stringWorld = "World!";

Console.WriteLine("Сравниваемые строки: \"" + stringHello + "\" и \"" + stringWorld + "\nРезультат сравнения: " + String.Compare(stringHello, stringWorld));

// Если первая строка по алфавиту стоит выше второй, то возвращается число меньше нуля. В противном случае возвращается число больше нуля.

// И третий случай - если строки равны, то возвращается число 0.

string string1 = "Lorem ",

string2 = "ipsum ",

string3 = "dolor";

Console.WriteLine("Сцепление: " + string1 + string2 + string3);

string Lorem = "Lorem ipsum dolor sit amet.";

// Разделение строки на подстроки

Console.WriteLine("Разделение строки на подстроки:");

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

foreach (string c in words)

Console.WriteLine(c);

// Копирование строк

string copyOfString1 = string1;

// Выделение подстроки

Console.Write("Выделение подстроки: ");

for (int m = 0; m < 18; m++)

Console.Write(Lorem[m]);

Console.WriteLine(Lorem.Substring(18).ToUpper());

// Вставка подстроки в строку

string aaa = "aaaaa",

bbb = "bbbbb";

Console.WriteLine("Вставка подстроки в строку: " + aaa.Insert(2, bbb));

// Удаление подстроки

string num = "0123456789";

Console.WriteLine("Удаление подстроки: " + num.Remove(0, 5));

// Интерполирование строк

int one = 1,

two = 2;

Console.WriteLine($"Интерполирование строк: один - {one}, два - {two}");

// Пустые и null строки

string? nullString = null;

string emptyString = "";

if (String.IsNullOrEmpty(nullString))

Console.WriteLine("Строка пустая или заполнена null-значением");

Console.WriteLine(nullString ?? Lorem);

// StringBuilder

StringBuilder sb = new StringBuilder("Hello World!");

Console.WriteLine("\n---StringBuilder---\nНачальная строка: " + sb);

sb.Remove(0, 5);

sb.AppendFormat("!!");

sb.Insert(0, "Привет");

Console.WriteLine("Строка после форматирования: " + sb);

// МАССИВЫ

// Создание и вывод матрицы

int[,] arr = {

{1,10},

{2,20},

{3,30},

{4,40}

};

Console.WriteLine("Матрица:");

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

{

for (int n = 0; n < 2; n++)

Console.Write(arr[k, n] + "\t");

Console.WriteLine();

}

// Одномерный массив строк

string[] arrOfStr = { "Hello ", "World!" };

foreach (string smth in arrOfStr)

Console.WriteLine(smth);

Console.WriteLine("Длина массива строк: " + arrOfStr.Length);

Console.Write("Выберете индекс строки: ");

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

if (index != 0 && index != 1)

Console.WriteLine("Введены неверные данные!");

else

{

Console.Write("Введите строку: ");

arrOfStr[index] = Console.ReadLine();

Console.Write("Полученная строка: ");

foreach (string smth in arrOfStr)

Console.Write(smth);

Console.WriteLine();

}

// Ступенчатый массив

int[][] steppedArray = new int[3][];

steppedArray[0] = new int[2];

steppedArray[1] = new int[3];

steppedArray[2] = new int[4];

for (i = 0; i < steppedArray.Length; i++)

for (j = 0; j < steppedArray[i].Length; j++)

{

Console.Write($"Введите число с координатами ({i},{j}): ");

steppedArray[i][j] = Convert.ToInt32(Console.ReadLine());

}

Console.WriteLine("Полученный массив: ");

for (i = 0; i < steppedArray.Length; i++)

{

for (j = 0; j < steppedArray[i].Length; j++)

Console.Write(steppedArray[i][j] + "\t");

Console.WriteLine();

}

// Неявно типизированные строки и массивы

var varStr = "Hello World!";

var varArr = new[] { 1, 2, 3, 4, 5 };

// КОРТЕЖИ

(int, string, char, string, ulong) t = (2, "Lorem", 'h', "ipsum", 234);

Console.WriteLine($"Кортеж: {t.Item1}, {t.Item2}, {t.Item3}, {t.Item4}, {t.Item5}");

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

// Распаковка кортежей

(int q, string w, char e, string r, ulong y) = t;

int cortInt;

string cortStr1, cortStr2;

char cortChar;

ulong cortUlong;

(cortInt, cortStr1, cortChar, cortStr2, cortUlong) = t;

var (c1, c2, c3, c4, c5) = t;

// Не понял задание про (\_), возможно имелось в виду

int million = 1\_000\_000;

// ref???

// Сравнение кортежей

if ((0, 2) == (-1, 1))

Console.WriteLine("Сравниваемые кортежи равны");

else

Console.WriteLine("Сравниваемые кортежи не равны");

// Локальная функция, которая возвращает кортеж

static (int, int, int, char) func(int[] arr, string str)

{

int min = arr[0],

max = arr[0],

sum = 0;

foreach(int i in arr)

{

if (i > max)

max = i;

if (i < min)

min = i;

sum += i;

}

return (max, min, sum, str[0]);

}

var returnedCort = func(new int[]{1,2,3,4,5}, "Hello");

// checked/unchecked

static void checkedFunc()

{

int x = checked(int.MaxValue);

}

static void uncheckedFunc()

{

int x = unchecked(int.MaxValue);

}

checkedFunc();

uncheckedFunc();

}

}

}

# № 3 Проектирование типов. Классы

using System;

using System.Diagnostics.Tracing;

using System.Reflection.Metadata.Ecma335;

using System.Security.Cryptography.X509Certificates;

using System.Text.RegularExpressions;

namespace ЛР\_\_3

{

class Program

{

public partial class Airline

{

// Объявление свойств

static int counter;

private string place;

private int num;

private string time;

private string dayOfWeek;

private int \_ID { get; }

public int ID

{

get { return \_ID; }

}

private int \_Number { get; set; }

public int Number

{

get { return \_Number; }

set { \_Number = value; }

}

private int \_forSet { get; set; }

public int forSet

{

set { \_forSet = value; }

}

const double PI = Math.PI;

static int forStatic;

// Конуструктор с проверкой данных

public Airline(string Place, string Time, string DayOfWeek, int Num = 5)

{

Regex TimeRegex = new Regex(@"^([0-1]?[0-9]|2[0-3]):[0-5][0-9]$");

if (TimeRegex.Matches(Time).Count > 0)

{

switch (DayOfWeek)

{

case "Понедельник": dayOfWeek = DayOfWeek; break;

case "Вторник": dayOfWeek = DayOfWeek; break;

case "Среда": dayOfWeek = DayOfWeek; break;

case "Четверг": dayOfWeek = DayOfWeek; break;

case "Пятница": dayOfWeek = DayOfWeek; break;

case "Суббота": dayOfWeek = DayOfWeek; break;

case "Воскресенье": dayOfWeek = DayOfWeek; break;

default: return;

}

place = Place;

num = Num;

time = Time;

}

else return;

counter++;

}

// Конструктор с двуми параметрами

public Airline(int Num, string Place)

{

num = Num;

place = Place;

time = "10:00";

dayOfWeek = "Вторник";

}

// Конструктор без параметров

public Airline()

{

place = "Тбилиси";

num = 2;

time = "13:25";

dayOfWeek = "Суббота";

counter++;

}

// Статический конструктор

static Airline()

{

forStatic = 12;

counter++;

}

// Закрытый конструктор

private Airline(int Num)

{

\_ID = GetHashCode();

}

// Интерфейс

public string getPlace() { return place; }

public string getTime() { return time; }

public string getDayOfWeek() { return dayOfWeek; }

public int getNum() { return num; }

public int getForStatic() { return forStatic; }

public void Func(ref int val1, out int val2)

{

//val1 = 5;

Console.WriteLine(val1);

val2 = 10; // Обязательно

Console.WriteLine(val2);

}

public static void printCounter()

{

Console.WriteLine("Количество экземплятров класса: " + counter);

}

}

public partial class Airline

{

public void Write()

{

Console.WriteLine("Hello World!");

}

}

// Вывод экземпляра класса

static void print(Airline airline)

{ Console.WriteLine($"{airline.getNum()}\t{airline.getDayOfWeek()}\t{airline.getTime()}\t{airline.getPlace()}"); }

// Поиск рейсов по заданному параметру

// type - код параметра, по которому идет сортировка

// 0 - по назначению

// 1 - по дню недели

static void find(Airline[] airlines, string condition, byte type)

{

if (type == 0)

Console.WriteLine($"---Список рейсов для назначения: {condition}---");

else if (type == 1)

Console.WriteLine($"---Список рейсов для дня недели: {condition}---");

foreach (Airline airline in airlines)

{

if (type == 0 && airline.getPlace() == condition)

print(airline);

else if (type == 1 && airline.getDayOfWeek() == condition)

print(airline);

}

Console.WriteLine();

}

static void Main(string[] args)

{

Airline[] airlines = new Airline[5];

airlines[0] = new Airline(1, "Минск");

airlines[1] = new Airline();

airlines[2] = new Airline("Пекин", "21:50", "Четверг", 3);

airlines[3] = new Airline("Сочи", "8:45", "Пятница", 4);

airlines[4] = new Airline("Минск", "22:10", "Суббота");

find(airlines, "Суббота", 1);

find(airlines, "Минск", 0);

Console.WriteLine(airlines[0].getForStatic());

airlines[0].Number = 9;

Console.WriteLine(airlines[0].Number);

Console.WriteLine(airlines[0].ID);

airlines[0].forSet = 12;

//Console.WriteLine(airlines[0].forSet); - нет прав

int a = 10,

b = 3;

airlines[0].Func(ref a, out b);

airlines[0].Write();

}

}

}

# № 4 Перегрузка операций, методы расширения и вложенные типы

using System;

using System.Buffers;

using System.Diagnostics.SymbolStore;

using System.Runtime.CompilerServices;

using System.Runtime.Serialization.Formatters;

namespace ЛР\_\_4

{

class Program

{

static void Main(string[] args)

{

MyArray<int> arr1 = new MyArray<int>(3);

MyArray<int> arr2 = new MyArray<int>(3);

arr1[0] = 1;

arr1[1] = 2;

arr1[2] = 3;

arr2[0] = 1;

arr2[1] = 2;

arr2[2] = 3;

Console.WriteLine($"Включена ли 2 в первый массив: {2 > arr1}");

Console.WriteLine($"Включена ли 5 во второй массив: {5 > arr2}");

Console.WriteLine($"Не равны ли массивы: {arr1 != arr2}");

MyArray<int> arr = new MyArray<int>((arr1 + arr2).Size());

arr = arr1 + arr2;

Console.Write("Объединение массивов: ");

for (int i = 0; i < arr.Size(); i++)

Console.Write(arr[i] + " ");

arr.DeleteFirstFive();

Console.Write("\nОбъединенный массив после удаления пяти первых элементов: ");

for (int i = 0; i < arr.Size(); i++)

Console.Write(arr[i] + " ");

}

}

class Owner

{

public string name;

public string organization;

public uint id;

}

class MyArray<T>

{

private Owner person = new Owner { name = "Yegor", organization = "BELSTU", id = 0 };

private Date date = new Date {date = (DateTime.Now).ToString() };

private T[] array;

private uint size;

public T this[int index]

{

get

{

return array[index];

}

set

{

array[index] = value;

}

}

public MyArray(uint Size)

{

this.array = new T[Size];

size = Size;

}

public static double operator -(MyArray<T> arr1, MyArray<T> arr2)

{

T diff;

double sum1 = 0, sum2 = 0;

for (int i = 0; i < arr1.size; i++)

sum1 += Convert.ToDouble(arr1[i]);

for (int i = 0; i < arr2.size; i++)

sum2 += Convert.ToDouble(arr2[i]);

return sum1 - sum2;

}

public static bool operator >(T e, MyArray<T> arr)

{

for (int i = 0; i < arr.size; i++)

if (Convert.ToDouble(e) == Convert.ToDouble(arr[i]))

return true;

return false;

}

public static bool operator <(T e, MyArray<T> arr2)

{ return false; }

public static bool operator !=(MyArray<T> arr1, MyArray<T> arr2)

{

if (arr1.size != arr2.size)

return true;

for (int i = 0; i < arr1.size; i++)

if (Convert.ToDouble(arr1[i]) != Convert.ToDouble(arr2[i]))

return true;

return false;

}

public static bool operator ==(MyArray<T> arr1, MyArray<T> arr2)

{

if (arr1.size != arr2.size)

return false;

for (int i = 0; i < arr1.size; i++)

if (Convert.ToDouble(arr1[i]) != Convert.ToDouble(arr2[i]))

return false;

return true;

}

public static MyArray<T> operator +(MyArray<T> arr1, MyArray<T> arr2)

{

MyArray<T> arr = new MyArray<T>(arr1.size + arr2.size);

int i = 0;

for (; i < arr1.size; i++)

arr[i] = arr1[i];

for (; i < arr2.size + arr1.size; i++)

arr[i] = arr2[i-Convert.ToInt32(arr1.size)];

return arr;

}

public void VowelsRemove()

{

string vowels = "уеыаоэяиюeyuioa";

for(int i = 0; i < this.size; i++)

{

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

if (Convert.ToChar(this[i]) == vowels[j])

this.Delete(i);

}

}

public void DeleteFirstFive()

{

uint count = this.size > 5 ? 5 : this.size;

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

this.Delete(i);

}

private void Delete(int index)

{

for (int i = index, j = i + 1; i < this.size - 1; i++, j = i + 1)

this[i] = this[j];

//this[Convert.ToInt32(size) - 1] = null;

this.size--;

}

public uint Size() { return size; }

class Date

{

public string date;

}

}

static class StaticOperation

{

static public int Sum(this MyArray<int> arr)

{

int sum = 0;

for (int i = 0; i < arr.Size(); i++)

sum += arr[i];

return sum;

}

static public int Diff(this MyArray<int> arr)

{

int max = arr[0], min = arr[0];

for (int i = 0; i < arr.Size(); i++)

{

if (arr[i] > max)

max = arr[i];

if (arr[i] < min)

min = arr[i];

}

return max - min;

}

static public int Size(this MyArray<int?> arr)

{

int counter = 0;

while (arr[counter] != null)

counter++;

return --counter;

}

public static int CharCount(this string str, char c)

{

int counter = 0;

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

{

if (str[i] == c)

counter++;

}

return counter;

}

}

}

# № 5 Наследование, полиморфизм, абстрактные классы и интерфейсы

using System;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.Drawing;

using System.Runtime.ConstrainedExecution;

using System.Xml.Serialization;

namespace ЛР\_\_5

{

class Program

{

static void Main(string[] args)

{

Circle Circle1 = new Circle();

Circle1.printSize(5);

Circle1.printSize();

Console.WriteLine(Circle1.ToString());

List<ControlElement> CE = new List<ControlElement>();

CE.Add(new Checkbox());

CE.Add(new Radiobutton());

CE.Add(new Button());

Printer printer = new Printer();

foreach(ControlElement E in CE)

{

printer.IAmPrinting(E);

if (E is Radiobutton)

Console.WriteLine(" - RadioButton");

else if (E is Button)

Console.WriteLine(" - Button");

else if (E is Checkbox)

Console.WriteLine(" - Checkbox");

}

}

}

interface IControl

{

void show();

void input();

void resize();

}

class Figure : IControl

{

protected int size { get; set; }

private string color { get; set; }

public virtual void show()

{

Console.Write("Фигура: ");

}

public void resize()

{

Console.Write("Введите размер: ");

size = Convert.ToInt32(Console.ReadLine());

}

public void input()

{

color = Console.ReadLine();

}

public void printSize()

{

Console.WriteLine($"Размер: {size}");

}

public override string ToString()

{

return "Использован метод ToString()";

}

}

sealed class Circle : Figure

{

public override void show()

{

base.show();

Console.WriteLine("круг");

}

public void printSize(int Size)

{

Console.WriteLine($"Радиус круга: {Size}");

size = Size;

}

public override string ToString()

{

return "Использован метод ToString()";

}

}

sealed class Rect : Figure

{

public override void show()

{

base.show();

Console.WriteLine("прямоугольник");

}

}

interface IControleElement

{

void Click();

}

abstract class ControlElement

{

public abstract void Click();

}

class Checkbox : ControlElement, IControleElement

{

private int number { get; set; }

public override void Click()

{

number = Convert.ToInt32(Console.ReadLine());

}

}

class Radiobutton : ControlElement, IControleElement

{

private bool isPressed { get; set; } = false;

public override void Click()

{

isPressed = Convert.ToBoolean(Console.ReadLine());

}

}

class Button : ControlElement, IControleElement

{

private int count { get; set; } = 0;

public override void Click()

{

Console.WriteLine("Кликните на кнопку");

Console.ReadKey();

count++;

}

}

class Printer

{

public virtual void IAmPrinting(object someObj)

{

Console.Write(someObj.GetType());

}

}

}

# № 6 Структуры, перечисления, классы контейнеры и контроллеры

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using System.Dynamic;

using System.Runtime.CompilerServices;

namespace ЛР\_\_6

{

class Program

{

static void Main(string[] args)

{

//Circle Circle1 = new Circle();

//Circle1.printSize(5);

//Circle1.printSize();

//Console.WriteLine(Circle1.ToString());

//List<UI> Figures = new List<UI>();

//Figures.Add(Circle1);

//Figures.Add(new Circle() { size = 55.54 });

//Figures.Add(new Rect() { size = 23.2 });

//List<ControlElement> CE = new List<ControlElement>();

//CE.Add(new Checkbox());

//CE.Add(new Radiobutton());

//CE.Add(new Button());

//Printer printer = new Printer();

//foreach (ControlElement E in CE)

//{

// printer.IAmPrinting(E);

// if (E is Radiobutton)

// Console.WriteLine($" - {EControlElements.RadioButton}");

// else if (E is Button)

// Console.WriteLine($" - {EControlElements.Button}");

// else if (E is Checkbox)

// Console.WriteLine($" - {EControlElements.Checkbox}");

//}

UI UserIntarface = new UI();

UIController uiController = new UIController();

bool isWork = true;

UserIntarface.Add(new Checkbox());

UserIntarface.Add(new Radiobutton());

UserIntarface.Add(new Button());

UserIntarface.Add(new Circle(), 23.2);

UserIntarface.Add(new Circle(), 55.54);

UserIntarface.Add(new Rect(), 23.2);

do

{

switch (UserIntarface.menu())

{

case 1: uiController.PrintList(UserIntarface); break;

case 2: Console.WriteLine("Количество всех созданных элементов: " + uiController.ElementCounter(UserIntarface)); break;

case 3: Console.WriteLine("Общая площадь всех фигур: " + uiController.TotalArea(UserIntarface)); break;

case 4: isWork = false; break;

default: Console.WriteLine("Некорректно введенные данные!"); break;

}

} while (isWork);

}

}

enum EControlElements { RadioButton, Button, Checkbox };

struct Types

{

private string str;

private int integer;

}

public interface IControl

{

void show();

void input();

void resize();

}

public partial class Figure : IControl

{

public double size { get; set; }

private string color { get; set; }

public virtual void show()

{

Console.Write("Фигура: ");

}

public void resize()

{

Console.Write("Введите размер: ");

size = Convert.ToInt32(Console.ReadLine());

}

public void input()

{

color = Console.ReadLine();

}

public void printSize()

{

Console.WriteLine($"Площадь: {size}");

}

}

public sealed class Circle : Figure

{

public override void show()

{

base.show();

Console.WriteLine("круг");

}

public void printSize(int Size)

{

Console.WriteLine($"Площадь круга: {Size}");

size = Size;

}

}

public sealed class Rect : Figure

{

public override void show()

{

base.show();

Console.WriteLine("прямоугольник");

}

}

public interface IControleElement

{

void Click();

}

public abstract class ControlElement

{

public abstract void Click();

public bool isControlElement = true;

}

public class Checkbox : ControlElement, IControleElement

{

private int number { get; set; }

public override void Click()

{

number = Convert.ToInt32(Console.ReadLine());

}

}

public class Radiobutton : ControlElement, IControleElement

{

private bool isPressed { get; set; } = false;

public override void Click()

{

isPressed = Convert.ToBoolean(Console.ReadLine());

}

}

public class Button : ControlElement, IControleElement

{

private int count { get; set; } = 0;

public override void Click()

{

Console.WriteLine("Кликните на кнопку");

Console.ReadKey();

count++;

}

}

class Printer

{

public virtual void IAmPrinting(object someObj)

{

Console.Write(someObj.GetType());

}

}

public partial class UI

{

public List<object> Elements = new List<object>();

public byte menu()

{

Console.Write($"1. Вывести список всех кнопок\n" +

$"2. Вывести общее количество всех элементов\n" +

$"3. Найти общую площадь, всех фигур\n" +

$"4. Выход\n" +

$"$ ");

return Convert.ToByte(Console.ReadLine());

}

}

public class UIController

{

public void PrintList(UI UI)

{

foreach (object obj in UI.Elements)

try

{

if (((ControlElement)obj).isControlElement)

Console.WriteLine(obj);

}

catch (System.InvalidCastException) { continue; }

}

public int ElementCounter(UI UI)

{

return UI.Elements.Count;

}

public double TotalArea(UI UI)

{

double Area = 0;

foreach (object obj in UI.Elements)

{

try

{

Area += ((Figure)obj).size;

}

catch (System.InvalidCastException) { continue; }

}

return Area;

}

}

}

using Microsoft.VisualBasic.FileIO;

using System;

using System.Collections.Generic;

using System.Reflection.Metadata.Ecma335;

using System.Text;

namespace ЛР\_\_6

{

partial class UI

{

public void Add(Rect Figure, double Size)

{

Figure.size = Size;

Elements.Add(Figure);

}

public void Add(Circle Figure, double Size)

{

Figure.size = Size;

Elements.Add(Figure);

}

public void Add(Button CE)

{

Elements.Add(CE);

}

public void Add(Checkbox CE)

{

Elements.Add(CE);

}

public void Add(Radiobutton CE)

{

Elements.Add(CE);

}

public object this[int index]

{

get

{

return Elements[index];

}

}

public void Delete(int index)

{

Elements.Remove(index);

}

public void Print()

{

foreach (object obj in Elements)

Console.WriteLine(obj);

# № 7 Исключения

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

namespace ЛР\_\_7

{

class Program

{

static void Main(string[] args)

{

try

{

int[] x = new int[3];

x[3] = 5;

int zero = 0;

int res = 5 / zero;

throw new Exception("Ошибка №1");

throw new Exception("Ошибка №2");

}

catch(Exception ex)

{

new ConsoleLogger("ERROR", ex.Message);

new FileLogger("ERROR", ex.Message);

}

finally

{

Console.WriteLine("Сработал блок finally");

}

Console.WriteLine("Если Вы введете 0, то программа экстренно завершится");

Debug.Assert(Convert.ToByte(Console.ReadLine()) != 0);

UI UserIntarface = new UI();

UIController uiController = new UIController();

bool isWork = true;

UserIntarface.Add(new Checkbox());

UserIntarface.Add(new Radiobutton());

UserIntarface.Add(new Button());

UserIntarface.Add(new Circle(), 23.2);

UserIntarface.Add(new Circle(), 55.54);

UserIntarface.Add(new Rect(), 23.2);

do

{

switch (UserIntarface.menu())

{

case 1: uiController.PrintList(UserIntarface); break;

case 2: Console.WriteLine("Количество всех созданных элементов: " + uiController.ElementCounter(UserIntarface)); break;

case 3: Console.WriteLine("Общая площадь всех фигур: " + uiController.TotalArea(UserIntarface)); break;

case 4: isWork = false; break;

default: Console.WriteLine("Некорректно введенные данные!"); break;

}

} while (isWork);

}

}

enum EControlElements { RadioButton, Button, Checkbox };

struct Types

{

private string str;

private int integer;

}

public interface IControl

{

void show();

void input();

void resize();

}

public partial class Figure : IControl

{

public double size { get; set; }

private string color { get; set; }

public virtual void show()

{

Console.Write("Фигура: ");

}

public void resize()

{

Console.Write("Введите размер: ");

try

{

size = Convert.ToInt32(Console.ReadLine());

if (size < 0)

throw (new Exception("Размер не может быть отрицательным!"));

}

catch(Exception ex)

{

new ConsoleLogger("ERROR", ex.Message);

new FileLogger("ERROR", ex.Message);

return;

}

}

public void input()

{

color = Console.ReadLine();

}

public void printSize()

{

Console.WriteLine($"Площадь: {size}");

}

}

public sealed class Circle : Figure

{

public override void show()

{

base.show();

Console.WriteLine("круг");

}

public void printSize(int Size)

{

Console.WriteLine($"Площадь круга: {Size}");

size = Size;

}

}

public sealed class Rect : Figure

{

public override void show()

{

base.show();

Console.WriteLine("прямоугольник");

}

}

public interface IControlElement

{

void Click();

}

public abstract class ControlElement

{

public abstract void Click();

public bool isControlElement = true;

}

public class Checkbox : ControlElement, IControlElement

{

private int number { get; set; }

public override void Click()

{

number = Convert.ToInt32(Console.ReadLine());

}

}

public class Radiobutton : ControlElement, IControlElement

{

private bool isPressed { get; set; } = false;

public override void Click()

{

isPressed = Convert.ToBoolean(Console.ReadLine());

}

}

public class Button : ControlElement, IControlElement

{

private int count { get; set; } = 0;

public override void Click()

{

Console.WriteLine("Кликните на кнопку");

Console.ReadKey();

count++;

}

}

class Printer

{

public virtual void IAmPrinting(object someObj)

{

Console.Write(someObj.GetType());

}

}

public class UI

{

public List<object> Elements = new List<object>();

public byte menu()

{

Console.Write($"1. Вывести список всех кнопок\n" +

$"2. Вывести общее количество всех элементов\n" +

$"3. Найти общую площадь, всех фигур\n" +

$"4. Выход\n" +

$"$ ");

try

{

byte choice = Convert.ToByte(Console.ReadLine());

if (choice > 4 || choice < 1)

throw (new Exception("Неверно введенные данные!"));

else return choice;

}

catch(Exception ex)

{

new ConsoleLogger("ERROR", ex.Message);

new FileLogger("ERROR", ex.Message);

return 0;

}

}

public void Add(Rect Figure, double Size)

{

Figure.size = Size;

Elements.Add(Figure);

}

public void Add(Circle Figure, double Size)

{

Figure.size = Size;

Elements.Add(Figure);

}

public void Add(Button CE)

{

Elements.Add(CE);

}

public void Add(Checkbox CE)

{

Elements.Add(CE);

}

public void Add(Radiobutton CE)

{

Elements.Add(CE);

}

public object this[int index]

{

get

{

return Elements[index];

}

}

public void Delete(int index)

{

Elements.Remove(index);

}

public void Print()

{

foreach (object obj in Elements)

Console.WriteLine(obj);

}

}

public class UIController

{

public void PrintList(UI UI)

{

foreach (object obj in UI.Elements)

try

{

if (((ControlElement)obj).isControlElement)

Console.WriteLine(obj);

}

catch (InvalidCastException ex)

{

new ConsoleLogger("ERROR", ex.Message);

new FileLogger("ERROR", ex.Message);

continue;

}

}

public int ElementCounter(UI UI)

{

return UI.Elements.Count;

}

public double TotalArea(UI UI)

{

double Area = 0;

foreach (object obj in UI.Elements)

{

try

{

Area += ((Figure)obj).size;

}

catch (InvalidCastException ex)

{

new ConsoleLogger("INFO", ex.Message);

new FileLogger("INFO", ex.Message);

continue;

}

}

return Area;

}

}

class Logger

{

protected string Message { get; set; }

}

class FileLogger : Logger

{

public FileLogger(string type, string msg)

{

DateTime now = DateTime.Now;

Message = now + " " + type + ": " +msg + '\n';

File.AppendAllText("D:\\Учеба\\ООП\\ЛР №7\\log.txt", Message);

}

}

class ConsoleLogger : Logger

{

public ConsoleLogger(string type, string msg)

{

Message = DateTime.Now + " " + type + ": " + msg;

Console.WriteLine(Message);

}

}

}

# № 8 Обобщения

using System;

using System.Collections.Generic;

using System.IO;

namespace ЛР\_\_8

{

class Program

{

static void Main(string[] args)

{

CollectionType<int> arr1 = new CollectionType<int>(3);

CollectionType<int> arr2 = new CollectionType<int>(3);

arr1[0] = 1;

arr1[1] = 2;

arr1[2] = 3;

arr2[0] = 1;

arr2[1] = 2;

arr2[2] = 3;

arr1.SaveAsFile();

arr2.SaveAsFile();

}

interface IMain<T> where T : struct

{

public void Add(T index);

public void Delete(T index);

public void Print(T index);

}

class Owner

{

public string name;

public string organization;

public uint id;

}

class CollectionType<T> : IMain<int>

{

private Owner person = new Owner { name = "Yegor", organization = "BELSTU", id = 0 };

private Date date = new Date { date = (DateTime.Now).ToString() };

private T[] array;

private uint size;

public T this[int index]

{

get

{

return array[index];

}

set

{

array[index] = value;

}

}

public CollectionType(uint Size)

{

array = new T[Size];

size = Size;

}

public static double operator -(CollectionType<T> arr1, CollectionType<T> arr2)

{

T diff;

double sum1 = 0, sum2 = 0;

for (int i = 0; i < arr1.size; i++)

sum1 += Convert.ToDouble(arr1[i]);

for (int i = 0; i < arr2.size; i++)

sum2 += Convert.ToDouble(arr2[i]);

return sum1 - sum2;

}

public static bool operator >(T e, CollectionType<T> arr)

{

for (int i = 0; i < arr.size; i++)

if (Convert.ToDouble(e) == Convert.ToDouble(arr[i]))

return true;

return false;

}

public static bool operator <(T e, CollectionType<T> arr2)

{ return false; }

public static bool operator !=(CollectionType<T> arr1, CollectionType<T> arr2)

{

if (arr1.size != arr2.size)

return true;

for (int i = 0; i < arr1.size; i++)

if (Convert.ToDouble(arr1[i]) != Convert.ToDouble(arr2[i]))

return true;

return false;

}

public static bool operator ==(CollectionType<T> arr1, CollectionType<T> arr2)

{

if (arr1.size != arr2.size)

return false;

for (int i = 0; i < arr1.size; i++)

if (Convert.ToDouble(arr1[i]) != Convert.ToDouble(arr2[i]))

return false;

return true;

}

public static CollectionType<T> operator +(CollectionType<T> arr1, CollectionType<T> arr2)

{

CollectionType<T> arr = new CollectionType<T>(arr1.size + arr2.size);

int i = 0;

for (; i < arr1.size; i++)

arr[i] = arr1[i];

for (; i < arr2.size + arr1.size; i++)

arr[i] = arr2[i - Convert.ToInt32(arr1.size)];

return arr;

}

public void Delete(int index)

{

for (int i = index, j = i + 1; i < size - 1; i++, j = i + 1)

this[i] = this[j];

//this[Convert.ToInt32(size) - 1] = null;

size--;

}

public void Add(int index)

{

return;

}

public void Print(int index)

{

return;

}

class Date

{

public string date;

}

public void SaveAsFile()

{

string Message = $"Владелец: {person.name}\nID владельца: {person.id}\nОрганизация: {person.organization}\nМассив: " ;

foreach (T a in array)

Message += a + " ";

Message += "\n---------------------------------------------------------\n";

File.AppendAllText("D:\\Учеба\\ООП\\ЛР №8\\INFO.txt", Message);

}

}

class Figure

{

public int size { get; set; }

public string color { get; set; }

}

class FiguresControl<T> where T : Figure

{

public virtual void show()

{

Console.Write("Фигура: ");

}

public void resize(T Figure)

{

Console.Write("Введите размер: ");

Figure.size = Convert.ToInt32(Console.ReadLine());

}

public void input(T Figure)

{

Figure.color = Console.ReadLine();

}

public void printSize(T Figure)

{

Console.WriteLine($"Размер: {Figure.size}");

}

public override string ToString()

{

return "Использован метод ToString()";

}

}

}

}

# №9 Делегаты, события и лямбда выражения

using System;

namespace ЛР\_\_9

{

class Program

{

static void Main(string[] args)

{

User[] Users = new User[3];

Users[0] = new User();

Users[1] = new User();

Users[2] = new User();

Users[0].Move += (user, bias) => Console.WriteLine($"Пользователь {user} сдвинул объект на {bias} метров.");

Users[0].Squeeze += (user, factor) => Console.WriteLine($"Пользователь {user} сжал объект. Коэффициент сжатия: {factor}.");

Users[1].Move += (user, bias) => Console.WriteLine($"Пользователь {user} сдвинул объект на {bias} метров.");

Random rnd = new Random();

foreach(User u in Users)

{

u.move(u, rnd.NextDouble());

u.squeeze(u, rnd.NextDouble());

}

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

// Обработка строки

string String = "Hello World!";

StringProcessing str = new StringProcessing();

Func<string, string> StringProcessingDelegate = str => str.ToLower();

String = StringProcessingDelegate(String);

Console.WriteLine(String);

StringProcessingDelegate += str => str.ToUpper();

String = StringProcessingDelegate(String);

Console.WriteLine(String);

StringProcessingDelegate += str.RemovePunctuationMarks;

String = StringProcessingDelegate(String);

Console.WriteLine(String);

StringProcessingDelegate += str.Trim;

String = StringProcessingDelegate(String);

Console.WriteLine(String);

Func<string, char, string> StringProcessingWithoutParameters = (str, ch) => str += ch;

String = StringProcessingWithoutParameters(String, 'w');

Console.WriteLine(String);

}

class User

{

public delegate void Operation(User user, double param);

public event Operation Move;

public event Operation Squeeze;

public string str { get; set; }

public void move(User user, double bias)

{

if (Move == null)

Console.WriteLine("Вызвано событие Move");

else

Move(user, bias);

}

public void squeeze(User user, double factor)

{

if (Squeeze == null)

Console.WriteLine("Вызвано событие Squeeze");

else

Squeeze(user, factor);

}

}

class StringProcessing

{

public string RemovePunctuationMarks(string str)

{

string PunctuatinMarks = ",.;:-!";

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

{

foreach (char PunctuationMark in PunctuatinMarks)

if (str[i] == PunctuationMark)

str = str.Remove(i, 1);

}

return str;

}

public string Trim(string str)

{

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

if (str[i] == ' ' && str[i + 1] == ' ')

str = str.Remove(i + 1, 1);

return str;

}

}

}

}

# № 10 Коллекции

using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

namespace ЛР\_\_10

{

class Program

{

static void Main(string[] args)

{

Book book = new Book();

book.Add(0, "Hello World!");

book.Add(0, "Lorem Ipsum dolor");

book.Add(0, "Sit amet consueror");

book.Add(0, "Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium");

book.PrintBook();

book.Remove(2);

book.PrintBook();

book.PageSearchByNumber(2);

book.PageSearchByText("Hello World!");

BookDictionary DictionaryBook = new BookDictionary();

book.CloneToDictionary(ref DictionaryBook);

DictionaryBook.Print();

Console.WriteLine($"Вторая страница: {DictionaryBook[2]}");

ObservableCollection<Book> BookInObservableCollection = new ObservableCollection<Book>();

BookInObservableCollection.CollectionChanged += Book\_CollectionChanged;

BookInObservableCollection.Add(book);

BookInObservableCollection.Add(new Book());

BookInObservableCollection.RemoveAt(1);

}

private static void Book\_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)

{

switch (e.Action)

{

case NotifyCollectionChangedAction.Add:

Console.WriteLine("Добавлен новый объект");

break;

case NotifyCollectionChangedAction.Remove:

Console.WriteLine("Удален объект");

break;

}

}

class Book : IDictionary<int, string>

{

public ICollection<string> Values { get; }

public ICollection<int> Keys { get; }

public struct Page

{

public string \_Text { get; private set; }

public int \_Number { get; private set; }

public Page(string Text, int Number)

{

\_Text = Text;

\_Number = Number;

}

}

private List<Page> BookPages = new List<Page>();

public int Count { get; private set; } = 0;

public bool IsReadOnly { get; }

string IDictionary<int, string>.this[int key] { get => BookPages[key].\_Text; set => throw new NotImplementedException(); }

public void Add(int Number, string Text)

{

BookPages.Add(new Page(Text, ++Count));

}

public bool Remove(int Number)

{

if (BookPages.Remove(BookPages[--Number]))

{

Console.WriteLine("Успешно удалено.");

Count--;

for (int i = Number; i < Count; i++)

{

Page page = new Page(BookPages[i].\_Text, i + 1);

BookPages[i] = page;

}

return true;

}

else

{

Console.WriteLine("Ошибка удаления!");

return false;

}

}

public void PageSearchByText(string Text)

{

foreach (Page page in BookPages)

{

if (page.\_Text == Text)

{

Console.WriteLine($"Страница найдена!\nНомер: {page.\_Number}\nТекст: {page.\_Text}");

return;

}

}

Console.WriteLine("Страница не найдена!\n");

}

public void PageSearchByNumber(int Number)

{

try

{

Console.WriteLine($"Страница найдена!\nНомер: {BookPages[Number - 1].\_Number}\nТекст: {BookPages[Number - 1].\_Text}\n");

}

catch (Exception)

{

Console.WriteLine("Страница не найдена!\n");

}

}

public void PrintBook()

{

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

foreach (Page page in BookPages)

Console.WriteLine($"Номер страницы: {page.\_Number}\nТекст: {page.\_Text}");

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

}

public void RemoveN(int start, int n)

{

BookPages.RemoveRange(start, n);

}

public void Insert(Page page, int index)

{

BookPages.Insert(index, page);

}

public void AddRange(Page[] pages)

{

BookPages.AddRange(pages);

}

public void CloneToDictionary(ref BookDictionary bookDictionary)

{

foreach (Page page in BookPages)

bookDictionary.Add(page.\_Number, page.\_Text);

}

bool IDictionary<int, string>.ContainsKey(int key)

{

throw new NotImplementedException();

}

bool IDictionary<int, string>.TryGetValue(int key, out string value)

{

throw new NotImplementedException();

}

void ICollection<KeyValuePair<int, string>>.Add(KeyValuePair<int, string> item)

{

throw new NotImplementedException();

}

void ICollection<KeyValuePair<int, string>>.Clear()

{

throw new NotImplementedException();

}

bool ICollection<KeyValuePair<int, string>>.Contains(KeyValuePair<int, string> item)

{

throw new NotImplementedException();

}

void ICollection<KeyValuePair<int, string>>.CopyTo(KeyValuePair<int, string>[] array, int arrayIndex)

{

throw new NotImplementedException();

}

bool ICollection<KeyValuePair<int, string>>.Remove(KeyValuePair<int, string> item)

{

throw new NotImplementedException();

}

IEnumerator<KeyValuePair<int, string>> IEnumerable<KeyValuePair<int, string>>.GetEnumerator()

{

throw new NotImplementedException();

}

IEnumerator IEnumerable.GetEnumerator()

{

throw new NotImplementedException();

}

}

class BookDictionary

{

Dictionary<int, string> BookPages = new Dictionary<int, string>();

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

public void Add(int key, string text)

{

BookPages.Add(key, text);

Keys.Add(key);

}

public bool Print()

{

if (Keys == null)

return false ;

foreach (int key in Keys)

{

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

Console.WriteLine($"Номер страницы: {key}\nТекст: {BookPages[key]}");

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

}

return true;

}

public string PrintPage(int number)

{

return BookPages[number];

}

public string this[int number]

{

get => BookPages[number];

}

}

}

# № 11 LINQ to Object

using System;

using System.Collections.Generic;

using System.Linq;

namespace ЛР\_\_11

{

class Program

{

static void Main(string[] args)

{

List<string> months = new List<string>

{

"Январь",

"Февраль",

"Март",

"Апрель",

"Май",

"Июнь",

"Июль",

"Август",

"Сентрябрь",

"Октябрь",

"Ноябрь",

"Декабрь"

};

Console.WriteLine("Строки из 7 символов:");

var answer = months.Where(x => x.Count() == 7);

foreach (var a in answer)

Console.WriteLine(a);

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

Console.WriteLine("Летние месяцы:");

answer = months.SkipWhile(x => x != "Июнь").Reverse().SkipWhile(x => x != "Август").Reverse();

foreach (var a in answer)

Console.WriteLine(a);

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

Console.WriteLine("Зимние месяцы:");

List<string> ans = new List<string>(months);

ans.RemoveRange(2, 9);

foreach (var a in ans)

Console.WriteLine(a);

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

Console.WriteLine("Сортировка по алфавиту:");

answer = months.OrderBy(x => x);

foreach (var a in answer)

Console.WriteLine(a);

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

Console.WriteLine("Содержащие букву \"а\" и размером не менее 4-х символов:");

answer = months.Where(x => x.Contains('а')).Where(x => x.Length >= 4);

foreach (var a in answer)

Console.WriteLine(a);

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

List<Airline> Airline = new List<Airline>();

Airline.Add(new Airline("Пекин", "21:50", "Четверг"));

Airline.Add(new Airline("Сочи", "8:45", "Пятница"));

Airline.Add(new Airline("Москва", "8:15", "Вторник"));

Airline.Add(new Airline("Минск", "9:10", "Среда"));

Airline.Add(new Airline("Париж", "12:32", "Суббота"));

Airline.Add(new Airline("Париж", "18:46", "Среда"));

Airline.Add(new Airline("Мюнхен", "20:20", "Четверг"));

Airline.Add(new Airline("Париж", "7:20", "Пятница"));

var ParisRaces = Airline.Where(x => x.getPlace().Contains("Париж"));

Console.WriteLine("Рейсы в Париж:");

foreach (var race in ParisRaces)

race.print();

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

Console.WriteLine("Количество рейсов в среду: {0}", Airline.Where(x => x.getDayOfWeek().Contains("Среда")).Count());

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

Console.WriteLine("Рейс, который вылетает раньше всех в среду:");

var MinTime = Airline.Where(x => x.getDayOfWeek().Contains("Среда")).Min(x => x.TimeIndex);

var MinTimeObject = Airline.FirstOrDefault(a => a.TimeIndex == MinTime);

MinTimeObject.print();

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

Console.WriteLine("Рейс, который вылетает в среду или пятницу раньше всех:");

var MaxTime = Airline.Where(x => x.getDayOfWeek().Contains("Среда") || x.getDayOfWeek().Contains("Пятница")).Max(x => x.TimeIndex);

var MaxTimeObject = Airline.FirstOrDefault(a => a.TimeIndex == MaxTime);

MaxTimeObject.print();

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

Console.WriteLine("Список рейсов, упорядоченный по времени вылета: ");

var OrderByTime = Airline.OrderBy(x => x.TimeIndex);

foreach (var race in OrderByTime)

race.print();

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

// Сложный запрос

Console.WriteLine("Сложный запрос");

List<int> Numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100 };

Console.WriteLine(Numbers.OrderByDescending(x => x).Skip(2).Append(55).Reverse().Aggregate((x, y) => x + y));

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

// Join

Console.WriteLine("Join:");

Person magnus = new Person { Name = "Hedlund, Magnus" };

Person terry = new Person { Name = "Adams, Terry" };

Person charlotte = new Person { Name = "Weiss, Charlotte" };

Pet barley = new Pet { Name = "Barley", Owner = terry };

Pet boots = new Pet { Name = "Boots", Owner = terry };

Pet whiskers = new Pet { Name = "Whiskers", Owner = charlotte };

Pet daisy = new Pet { Name = "Daisy", Owner = magnus };

List<Person> people = new List<Person> { magnus, terry, charlotte };

List<Pet> pets = new List<Pet> { barley, boots, whiskers, daisy };

var query = people.Join(pets, person => person, pet => pet.Owner, (person, pet) => new { OwnerName = person.Name, Pet = pet.Name });

foreach (var obj in query)

{

Console.WriteLine("{0} - {1}", obj.OwnerName, obj.Pet);

}

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

}

class Person

{

public string Name { get; set; }

}

class Pet

{

public string Name { get; set; }

public Person Owner { get; set; }

}

}

}

# № 12 Рефлексия

using System;

using System.IO;

using System.Reflection;

using System.Collections.Generic;

namespace ЛР\_\_12

{

public static class Reflector

{

public static string Path = "D:\\Учеба\\ООП\\ЛР №12\\";

public static void AssemblyName(object obj)

{

Type type = obj.GetType();

string Info = "Имя сборки: " + type.Name + '\n' + "--------------------------\n";

File.AppendAllText(Path + type.Name + ".txt", Info);

}

public static void PublicContructors(object obj)

{

Type type = obj.GetType();

string Info = "Есть ли публичные конструкторы: ";

var ConstructorInfo = type.GetConstructors();

if (ConstructorInfo.Length > 0)

Info += "да\n";

else

Info += "нет\n";

Info += "--------------------------\n";

File.AppendAllText(Path + type.Name + ".txt", Info);

}

public static List<string> PublicMethods(object obj)

{

string PublicMethodsString = "Публичные методы: \n";

Type type = obj.GetType();

List<string> PublicMethodsList = new List<string>();

var AllMethods = type.GetMethods();

foreach (var MI in AllMethods)

{

PublicMethodsList.Add(MI.Name);

PublicMethodsString += MI.Name + "\n";

}

PublicMethodsString += "--------------------------\n";

if (PublicMethodsList.Count == 0)

File.AppendAllText(Path + type.Name + ".txt", "Публичные методы: нет\n" +

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

else

File.AppendAllText(Path + type.Name + ".txt", PublicMethodsString);

return PublicMethodsList;

}

public static List<string> FieldsAndPropertiesInfo(object obj)

{

Type type = obj.GetType();

string Info = "Поля:\n";

List<string> FieldsAndProperties = new List<string>();

var FieldsArr = type.GetFields();

foreach (FieldInfo FI in FieldsArr)

{

FieldsAndProperties.Add(FI.Name);

Info += FI.Name + '\n';

}

if (Info == "Поля:\n")

Info = "Поля: нет\n";

Info += "--------------------------\n";

var Properties = type.GetProperties();

Info += "Свойства: \n";

var InfoFields = Info;

foreach (var PI in Properties)

{

FieldsAndProperties.Add(PI.Name);

Info += PI.Name + '\n';

}

if (Info == InfoFields)

Info = InfoFields.Remove(InfoFields.LastIndexOf('\n')) + " нет";

Info += "--------------------------\n";

File.AppendAllText(Path + type.Name + ".txt", Info);

return FieldsAndProperties;

}

public static List<string> Interfaces(object obj)

{

Type type = obj.GetType();

string Info = "Реализованные интерфейсы:\n";

List<string> InterfacesList = new List<string>();

var Interfaces = type.GetInterfaces();

foreach (var I in Interfaces)

{

InterfacesList.Add(I.Name);

Info += I.Name + '\n';

}

Info += "--------------------------\n";

if (InterfacesList.Count == 0)

File.AppendAllText(Path + type.Name + ".txt", "Реализованные интерфейсы: нет\n--------------------------\n");

else

File.AppendAllText(Path + type.Name + ".txt", Info);

return InterfacesList;

}

public static void MethodsByParameterType(object obj, Type Parameter)

{

Type type = obj.GetType();

string Info = "Методы с типом параметра " + Parameter.Name + ":\n";

string InfoStart = Info;

var AllMethods = type.GetMethods();

foreach (var MI in AllMethods)

{

var MIParameters = MI.GetParameters();

foreach (var PI in MIParameters)

{

if (PI.ParameterType == Parameter)

{

Info += MI.Name + '\n';

break;

}

}

}

Info += "--------------------------\n";

if (InfoStart != Info)

File.AppendAllText(Path + type.Name + ".txt", Info);

else

File.AppendAllText(Path + type.Name + ".txt", "Методы с типом параметра " + Parameter.Name + ": нет" + "--------------------------\n");

}

public static object? Invoke(object obj, string MethodName, object?[]? Parameters)

{

Type type = obj.GetType();

var Method = type.GetMethod(MethodName);

var Result = Method.Invoke(obj, Parameters);

return Result;

}

public static object? Create<T>(object?[]? Params)

{

Type type = typeof(T);

var Constructors = type.GetConstructors();

var Parameters = Constructors[0].GetParameters();

object? Result = null;

if (Params == null)

{

foreach (var Constructor in Constructors)

{

if (Constructor.GetParameters().Length == 0)

{

Result = Constructor.Invoke(null);

break;

}

}

}

else

{

foreach (var Constructor in Constructors)

{

if (Constructor.GetParameters().Length == Params.Length)

{

Result = Constructor.Invoke(Params);

break;

}

}

}

return Result;

}

}

}

# № 13 Работа с потоковыми классами и файловой системой

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace ЛР\_\_13

{

static class AYVDirInfo

{

public static void Info(string path)

{

try

{

DirectoryInfo directory = new DirectoryInfo(path);

Console.WriteLine("Количество файлов: " + directory.GetFiles().Length);

Console.WriteLine("Время создания: " + directory.CreationTime);

Console.WriteLine("Количество поддиректорий: " + directory.GetDirectories().Length);

Console.WriteLine("Родительская директория: " + directory.Parent);

}

catch(Exception ex)

{

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

}

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace ЛР\_\_13

{

static class AYVDiskInfo

{

public static void Info() {

foreach (var drive in DriveInfo.GetDrives())

{

try

{

Console.WriteLine("Имя диска: " + drive.Name);

Console.WriteLine("Файловая система: " + drive.DriveFormat);

Console.WriteLine("Тип диска: " + drive.DriveType);

Console.WriteLine("Общий объем свободного места (в байтах): " + drive.TotalFreeSpace);

Console.WriteLine("Размер диска (в байтах): " + drive.TotalSize);

Console.WriteLine("Метка тома диска: " + drive.VolumeLabel);

}

catch (Exception ex)

{

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

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace ЛР\_\_13

{

static class AYVFileInfo

{

public static void Info(string path)

{

try

{

FileInfo file = new FileInfo(path);

Console.WriteLine("Польный путь: " + file.FullName);

Console.WriteLine("Размер:" + file.Length);

Console.WriteLine("Расширение: " + file.Extension);

Console.WriteLine("Имя: " + file.Name);

Console.WriteLine("Дата создания: " + File.GetCreationTime(path));

Console.WriteLine("Дата изменения: " + File.GetLastWriteTime(path));

}

catch(Exception ex)

{

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

}

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace ЛР\_\_13

{

static class AYVFileManager

{

public static void AYVInspector()

{

Directory.CreateDirectory("AYVInspect");

var drive = DriveInfo.GetDrives()[1];

string Message = "Список файлов и папок:\n";

Message += FilesAndDirectories(drive.Name);

File.AppendAllText("AYVInspect/Disk" + drive.Name.Remove(1) + "Info.txt", Message);

File.Copy("AYVInspect/Disk" + drive.Name.Remove(1) + "Info.txt", "AYVInspect/Disk" + drive.Name.Remove(1) + "Info[1].txt");

File.Delete("AYVInspect/Disk" + drive.Name.Remove(1) + "Info.txt");

}

private static string FilesAndDirectories(string path)

{

string Message = "";

List<string> ls = GetRecursFiles(path);

foreach (string fname in ls)

{

Message += fname;

}

return Message;

}

private static List<string> GetRecursFiles(string start\_path)

{

List<string> ls = new List<string>();

try

{

string[] folders = Directory.GetDirectories(start\_path);

foreach (string folder in folders)

{

ls.Add("Папка: " + folder + "\n");

ls.AddRange(GetRecursFiles(folder));

AYVLog.Log(folder, "Чтение имени папки");

}

string[] files = Directory.GetFiles(start\_path);

foreach (string filename in files)

{

ls.Add("Файл: " + filename + "\n");

AYVLog.Log(filename, "Чтение имени файла");

}

}

catch (System.Exception e)

{

Console.WriteLine("Error: " + e.Message);

}

return ls;

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace ЛР\_\_13

{

static class AYVLog

{

private static string path = "AYVLog.txt";

private static string path1 = "AYVLog1.txt";

public static void Log(string FileName, string Message)

{

string msg = $"[{DateTime.Now}] {FileName}: {Message}";

using(StreamWriter sw = File.AppendText(path))

{

sw.WriteLineAsync(msg);

}

}

public static List<string>? Search(string SearchData)

{

using (StreamReader sr = new StreamReader(path))

{

string FileData = sr.ReadToEnd();

List<string> SearchedInfo = new List<string>();

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

{

for(int SearchDataIndex = 0, FileDataIndex = i; SearchDataIndex < SearchData.Length; SearchDataIndex++, FileDataIndex++)

{

if (FileData[FileDataIndex] == SearchData[SearchDataIndex])

{

if (SearchData.Length - 1 == SearchDataIndex)

{

while (FileData[FileDataIndex] != '[')

FileDataIndex--;

string Message = "";

while (FileData[FileDataIndex] != '\n')

{

Message += FileData[FileDataIndex];

FileDataIndex++;

}

SearchedInfo.Add(Message);

break;

}

else

continue;

}

else

break;

}

}

if (SearchedInfo.Count > 0)

return SearchedInfo;

else

return null;

}

}

public static List<string>? SearchByDay(string day)

{

if (Convert.ToInt32(day) < 0 || Convert.ToInt32(day) > 31)

return null;

if (day.Length == 1)

{

day += '0';

char[] arr = day.ToCharArray();

Array.Reverse(arr);

day = new string(arr);

}

List<string> SearchedData = new List<string>();

using(StreamReader sr = new StreamReader(path))

{

while(!sr.EndOfStream)

{

string Line = sr.ReadLine();

if (Line[1] == day[0] && Line[2] == day[1])

SearchedData.Add(Line);

}

}

if (SearchedData.Count == 0)

return null;

else

return SearchedData;

}

public static bool RemoverByTime(DateTime startTime, DateTime endTime)

{

if (startTime > endTime)

return false;

using (StreamReader sr = new StreamReader(path))

{

while (!sr.EndOfStream)

{

string Line = sr.ReadLine();

string date = "";

DateTime DT = new DateTime();

for (int i = 1; i < 20; i++)

date += Line[i];

try

{

DT = Convert.ToDateTime(date);

}

catch(Exception ex)

{

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

return false;

}

if (!(DT >= startTime && DT <= endTime))

{

using(StreamWriter sw = File.AppendText(path1))

{

sw.WriteLine(Line);

}

}

}

}

return true;

}

}

}

# № 14 Сериализация

using System;

using System.Collections.Generic;

using System.Xml;

using System.Linq;

using System.Xml.Linq;

namespace ЛР\_\_14

{

class Program

{

static void Main(string[] args)

{

List<Circle> cicrles = new List<Circle>();

cicrles.Add(new Circle(12, "Red"));

cicrles.Add(new Circle(2, "Purple"));

cicrles.Add(new Circle(5, "Pink"));

cicrles.Add(new Circle(33, "Orange"));

cicrles.Add(new Circle(52, "White"));

CustomSerializer.XMLSerializer(cicrles);

CustomSerializer.BinSerializer(cicrles);

CustomSerializer.JSONSerializer(cicrles);

//List<Circle> Returned = CustomDeserializer.BinDeserializer();

//List<Circle> Returned = CustomDeserializer.JSONDeserializer();

List<Circle> Returned = CustomDeserializer.XMLDeserializer();

foreach (var circle in Returned)

{

Console.WriteLine(circle);

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

}

// Выборка

XmlDocument xDoc = new XmlDocument();

xDoc.Load("Circle.xml");

XmlElement xRoot = xDoc.DocumentElement;

var element = xRoot.SelectSingleNode("Circle[color='Red']");

if (element != null)

Console.WriteLine("\n" + element.OuterXml);

element = xRoot.SelectSingleNode("Circle[size='52']");

if (element != null)

Console.WriteLine(element.OuterXml + "\n");

// Linq to XML

XDocument xDocument = new XDocument();

XElement Illustrator = new XElement("program");

Illustrator.Add(new XAttribute("name", "Illustrator"));

Illustrator.Add(new XElement("price", "1622 RUB"));

Illustrator.Add(new XElement("period", "12 months"));

XElement InCopy = new XElement("program");

InCopy.Add(new XAttribute("name", "InCopy"));

InCopy.Add(new XElement("price", "386 RUB"));

InCopy.Add(new XElement("period", "12 months"));

XElement Photoshop = new XElement("program");

Photoshop.Add(new XAttribute("name", "Photoshop"));

Photoshop.Add(new XElement("price", "800 RUB"));

Photoshop.Add(new XElement("period", "6 months"));

XElement Programs = new XElement("Programs");

Programs.Add(Illustrator);

Programs.Add(InCopy);

Programs.Add(Photoshop);

xDocument.Add(Programs);

xDocument.Save("AdobePrograms.xml");

XDocument xDocRoot = XDocument.Load("AdobePrograms.xml");

var items = from xe in xDocRoot.Element("Programs").Elements("program")

where xe.Attribute("name").Value == "InCopy" || xe.Element("period").Value == "6 months"

select new AdobeProgram

{

Name = xe.Attribute("name").Value,

Price = xe.Element("price").Value,

Period = xe.Element("period").Value

};

foreach(var item in items)

{

Console.WriteLine($"Name: {item.Name}\nPrice: {item.Price}\nPeriod: {item.Period}");

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

}

}

class AdobeProgram

{

public string Name { get; set; }

public string Price { get; set; }

public string Period { get; set; }

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Xml.Serialization;

using System.IO;

using System.Diagnostics;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text.Json;

namespace ЛР\_\_14

{

static class CustomSerializer

{

private static string subl = "D:\\SOFT\\Sublime Text 3\\sublime\_text.exe";

public static void XMLSerializer(List<Circle> circle)

{

XmlSerializer xml = new XmlSerializer(typeof(List<Circle>));

using(FileStream fs = new FileStream("Circle.xml", FileMode.OpenOrCreate))

{

xml.Serialize(fs, circle);

}

Process.Start(subl, "Circle.xml");

}

public static void BinSerializer(List<Circle> circle)

{

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = new FileStream("Circle.bin", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, circle);

}

Process.Start(subl, "Circle.bin");

}

public static void JSONSerializer(List<Circle> circle)

{

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

{

var json = Encoding.Default.GetBytes(JsonSerializer.Serialize(circle));

fs.Write(json, 0, json.Length);

}

Process.Start(subl, "Circle.json");

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Xml.Serialization;

using System.IO;

using System.Diagnostics;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text.Json;

namespace ЛР\_\_14

{

public static class CustomDeserializer

{

public static List<Circle> JSONDeserializer()

{

using (FileStream fs = File.OpenRead("Circle.json"))

{

byte[] array = new byte[fs.Length];

fs.Read(array, 0, array.Length);

string circle = Encoding.Default.GetString(array);

return JsonSerializer.Deserialize<List<Circle>>(circle);

}

}

public static List<Circle> XMLDeserializer()

{

XmlSerializer xml = new XmlSerializer(typeof(List<Circle>));

using (FileStream fs = File.OpenRead("Circle.xml"))

{

return xml.Deserialize(fs) as List<Circle>;

}

}

public static List<Circle> BinDeserializer()

{

BinaryFormatter formatter = new BinaryFormatter();

using (FileStream fs = File.OpenRead("Circle.bin"))

{

return formatter.Deserialize(fs) as List<Circle>;

}

}

}

}

# № 15 Работа с потоками выполнения

using System;

using System.Collections.Generic;

using System.Text;

using System.Diagnostics;

using System.IO;

using System.Reflection;

using System.Threading;

namespace ЛР\_\_15

{

public static class Threads

{

public static void ProcessInfo()

{

string Path = "ProcessInfo.txt";

using (StreamWriter sw = new StreamWriter(Path, false, Encoding.Default))

{

foreach (Process process in Process.GetProcesses())

{

try

{

sw.WriteLine($"Process name: {process.ProcessName}\n" +

$"ID: {process.Id}\n" +

$"Priority: {process.PriorityClass}\n" +

$"Start time: {process.StartTime}\n" +

$"Total processor time: {process.TotalProcessorTime}");

sw.WriteLine("-------------------------\n");

}

catch

{

continue;

}

}

}

}

public static void WorkWithDomain()

{

string Path = "DomainInfo.txt";

AppDomain CurrentDomain = AppDomain.CurrentDomain;

using (StreamWriter sw = new StreamWriter(Path, false, Encoding.Default))

{

sw.WriteLine($"Name: {CurrentDomain.FriendlyName}\n" +

$"Setup information: {CurrentDomain.SetupInformation}\n");

sw.WriteLine("All assemblies:");

foreach (var assembly in CurrentDomain.GetAssemblies())

sw.WriteLine(assembly.FullName);

}

// New domain

//AppDomain newDomain = AppDomain.CreateDomain("NewDomain");

//foreach (var assembly in CurrentDomain.GetAssemblies())

// newDomain.Load(assembly.FullName);

//AppDomain.Unload(newDomain);

}

private static Mutex mutex = new Mutex();

public static void ThreadingCounter()

{

Thread thread1 = new Thread(new ParameterizedThreadStart(Thread1));

Thread thread2 = new Thread(new ParameterizedThreadStart(Thread2));

Console.Write("Введите число N: ");

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

Console.WriteLine();

thread1.Start(n);

thread2.Start(n);

}

private static void Thread1(object N)

{

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

{

mutex.WaitOne();

Console.WriteLine("Thread 1 works: " + i);

Console.WriteLine($"Thread info:\n{ThreadInfo(Thread.CurrentThread)}");

Thread.Sleep(500);

mutex.ReleaseMutex();

}

}

private static void Thread2(object N)

{

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

{

mutex.WaitOne();

Console.WriteLine("Thread 2 works: " + i);

Console.WriteLine($"Thread info:\n{ThreadInfo(Thread.CurrentThread)}");

Thread.Sleep(500);

mutex.ReleaseMutex();

}

}

private static string ThreadInfo(Thread thread)

{

return $"Status: {thread.ThreadState}\n" +

$"Name: {thread.Name}\n" +

$"Priority: {thread.Priority}\n" +

$"ID: {thread.GetHashCode()}\n" +

$"-------------------------------------------";

}

public struct ThreadsSettings

{

public int N { get; set; }

public bool InTurn { get; set; }

public ThreadsSettings(int n, bool inTurn)

{

N = n;

InTurn = inTurn;

}

}

private static string path = "ThreadingSplitCounter.txt";

private static AutoResetEvent autoResetEvent = new AutoResetEvent(false);

public static void ThreadingSplitCounter(bool InTurn)

{

Thread thread1 = new Thread(new ParameterizedThreadStart(Thread3));

Thread thread2 = new Thread(new ParameterizedThreadStart(Thread4));

Console.Write("Введите число N: ");

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

Console.WriteLine();

thread1.Priority = ThreadPriority.Highest;

ThreadsSettings TS = new ThreadsSettings(n, InTurn);

thread1.Start(TS);

thread2.Start(TS);

}

private static void Thread3(object threadsSettings)

{

if (!((ThreadsSettings)threadsSettings).InTurn)

autoResetEvent.WaitOne();

for (int i = 1; i <= ((ThreadsSettings)threadsSettings).N; i+=2)

{

using(StreamWriter sw = new StreamWriter(path, true, Encoding.Default))

{

string Message = "Thread 1: " + i;

Console.WriteLine(Message);

sw.WriteLine(Message);

}

Thread.Sleep(1000);

if ((i == ((ThreadsSettings)threadsSettings).N - 1 || i == ((ThreadsSettings)threadsSettings).N) && ((ThreadsSettings)threadsSettings).InTurn == false)

autoResetEvent.Close();

else

{

autoResetEvent.Set();

autoResetEvent.WaitOne();

}

}

}

private static void Thread4(object threadsSettings)

{

if (((ThreadsSettings)threadsSettings).InTurn)

autoResetEvent.WaitOne();

for (int i = 2; i <= ((ThreadsSettings)threadsSettings).N; i += 2)

{

using (StreamWriter sw = new StreamWriter(path, true, Encoding.Default))

{

string Message = "Thread 2: " + i;

Console.WriteLine(Message);

sw.WriteLine(Message);

}

Thread.Sleep(300);

autoResetEvent.Set();

if(i == ((ThreadsSettings)threadsSettings).N)

autoResetEvent.Close();

else

autoResetEvent.WaitOne();

}

}

private static int Seconds = 1;

public static void TimerProgram()

{

Timer timer = new Timer(new TimerCallback(TimerThread), null, 0, 1000);

Thread.Sleep(9000);

}

private static void TimerThread(object obj)

{

Console.WriteLine("Секунд прошло с начала работы таймера: " + Seconds++);

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

using System.Threading;

using System.Linq;

namespace ЛР\_\_15

{

static class Warehouse

{

private static string Path = "Goods.txt";

private static int Carriage = 0;

private static Mutex mutex = new Mutex();

private struct MachineInfo

{

public int Time { get; set; }

public int Number { get; }

public MachineInfo(int time, int num)

{

Time = time;

Number = num;

}

}

public static void WarehouseUnloading()

{

new Thread(new ParameterizedThreadStart(Machine)).Start(new MachineInfo(100, 1));

new Thread(new ParameterizedThreadStart(Machine)).Start(new MachineInfo(300, 2));

new Thread(new ParameterizedThreadStart(Machine)).Start(new MachineInfo(150, 3));

}

private static void Machine(object MachineInfo)

{

while (Carriage != File.ReadLines(Path).Count())

{

mutex.WaitOne();

if(Carriage != File.ReadLines(Path).Count())

Console.WriteLine("Машина №" + ((MachineInfo)MachineInfo).Number + ": " + File.ReadLines(Path).ElementAt(Carriage++));

Thread.Sleep(((MachineInfo)MachineInfo).Time);

mutex.ReleaseMutex();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

namespace ЛР\_\_15

{

static class YouTubeWithoutPremiumIn2021

{

private static SemaphoreSlim Semaphore = new SemaphoreSlim(1);

private struct UseInfo

{

public int Time { get; }

public User user { get; }

public List<Channel> Channels { get; }

public bool IsWorks { get; set; }

public UseInfo(User u, List<Channel> channels, int time)

{

Time = time;

user = u;

Channels = channels;

IsWorks = true;

}

}

private static int TimeToExit = 5000;

public static void Work()

{

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

Users.Add(new User("Иван"));

Users.Add(new User("Петр"));

Users.Add(new User("Алексей"));

Users.Add(new User("Василий"));

List<Channel> Channels = new List<Channel>();

Channels.Add(new Channel("BBC"));

Channels.Add(new Channel("T-Series"));

Random rnd = new Random();

foreach(var u in Users)

{

Thread.Sleep(50);

new Thread(new ParameterizedThreadStart(ChannelUse)).Start(new UseInfo(u, Channels, rnd.Next(2000, 7000)));

}

}

private static void ChannelUse(object Info)

{

UseInfo useInfo = (UseInfo)Info;

Timer timer = new Timer(new TimerCallback(UserLeave), useInfo, TimeToExit, 0);

Random rnd = new Random();

while (true)

{

foreach (var channel in useInfo.Channels)

{

Semaphore.Wait();

if (useInfo.IsWorks == false)

goto END;

if (channel.Used == false)

{

timer.Change(TimeToExit + useInfo.Time + 10, 0);

channel.Used = true;

channel.UserName = useInfo.user.Name;

Console.WriteLine($"{channel.Name} используется пользователем {useInfo.user.Name}.");

Semaphore.Release();

Thread.Sleep(useInfo.Time);

Semaphore.Wait();

channel.Used = false;

channel.UserName = null;

Console.WriteLine($"{useInfo.user.Name} покинул канал {channel.Name}.");

timer.Change(TimeToExit, 0);

Semaphore.Release();

Thread.Sleep(rnd.Next(700, 3000));

Semaphore.Wait();

}

Semaphore.Release();

}

}

END:

return;

}

private static void UserLeave(object useInfo)

{

UseInfo Info = (UseInfo)useInfo;

Info.IsWorks = false;

Console.WriteLine(((UseInfo)useInfo).user.Name + " покинул сервис.");

}

}

}

# № 16 Платформа параллельных вычислений

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using System.IO;

using System.Diagnostics;

using System.Collections.Concurrent;

namespace ЛР\_\_16

{

public static class TPL

{

public static void ParallelVectorByNumber()

{

Stopwatch stopwatch = new Stopwatch();

const int SomeNumber = 182;

List<Task> Tasks = new List<Task>();

Random rnd = new Random();

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

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

{

Tasks.Add(new Task(() =>

{

for (int i = 0; i < 100000; i++)

Vector.Add(rnd.Next(1, 10000) \* SomeNumber);

}));

}

Console.WriteLine(Tasks[0].Status);

stopwatch.Start();

foreach (var task in Tasks)

task.Start();

Task.WaitAll(Tasks.ToArray());

stopwatch.Stop();

Console.WriteLine(Tasks[0].Status);

Console.WriteLine("Количество тактов при использовании Task: " + stopwatch.ElapsedTicks);

}

public static void VectorByNumber()

{

Stopwatch stopwatch = new Stopwatch();

const int SomeNumber = 182;

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

Random rnd = new Random();

stopwatch.Start();

for (int i = 0; i < 1000000; i++)

{

Vector.Add(rnd.Next(1, 10000) \* SomeNumber);

}

stopwatch.Stop();

Console.WriteLine("Количество тактов без использования Task: " + stopwatch.ElapsedTicks);

}

public static void ParallelVectorByNumberWithTocken()

{

const int SomeNumber = 182;

List<Task> Tasks = new List<Task>();

Random rnd = new Random();

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

CancellationTokenSource tokenSource = new CancellationTokenSource();

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

{

Tasks.Add(new Task(() =>

{

for (int i = 0; i < 1000000; i++)

Vector.Add(rnd.Next(1, 10000) \* SomeNumber);

}, tokenSource.Token));

}

foreach (var task in Tasks)

task.Start();

tokenSource.Cancel();

Console.WriteLine("Status: " + Tasks[5].Status);

}

public static void Continuation()

{

Task<double> task1 = Task.Run(() => { return Math.Pow(2, 64); });

Task<double> task2 = task1.ContinueWith(x => { return Math.Pow(264, 5) \* x.Result; });

Task task3 = task2.ContinueWith(x => { Console.WriteLine("Ответ: " + x.Result \* 12); });

task3.GetAwaiter().GetResult();

}

public static void ParallelFor()

{

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

Random rnd = new Random();

Stopwatch stopwatch = new Stopwatch();

stopwatch.Start();

Parallel.For(0, 1000000, i => { Arr.Add(rnd.Next(1, 1000)); });

stopwatch.Stop();

Console.WriteLine("Parallel.For: \t" + stopwatch.ElapsedTicks + " ticks");

stopwatch.Reset();

stopwatch.Start();

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

for (int i = 0; i < 1000000; i++)

Array.Add(rnd.Next(1, 1000));

stopwatch.Stop();

Console.WriteLine("For:\t\t" + stopwatch.ElapsedTicks + " ticks");

}

public static void ParallelInvoke()

{

Parallel.Invoke(() => { Console.WriteLine("First function"); }, () => { Console.WriteLine("Second function"); }, () => { Console.WriteLine("Third function"); });

}

private static Mutex mutex = new Mutex();

public static void Shop()

{

Dictionary<int?, string> Product = new Dictionary<int?, string>();

Product.Add(0, "Sofa");

Product.Add(1, "Armchair");

Product.Add(2, "Table");

Product.Add(3, "Poster");

Product.Add(4, "Stool");

Product.Add(5, "Linoleum");

Product.Add(6, "Bed");

Product.Add(7, "Wallpaper");

Product.Add(8, "Extension");

Product.Add(9, "Window");

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

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

{

Task.Run(() =>

{

Random rnd = new Random();

while (true)

{

Thread.Sleep(rnd.Next(1000, 5000));

mutex.WaitOne();

int Counter = 0;

Warehouse.Add(rnd.Next(0, 9));

Console.WriteLine("Warehouse: ");

foreach (var product in Warehouse)

{

Console.WriteLine(Product[product]);

Counter++;

}

if (Counter == 0)

{

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("Nothing");

Console.ResetColor();

}

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

mutex.ReleaseMutex();

Thread.Sleep(rnd.Next(1000, 5000));

}

});

}

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

{

Task.Run(() =>

{

Random rnd = new Random();

while (true)

{

Thread.Sleep(rnd.Next(1000, 5000));

foreach (var product in Warehouse)

{

mutex.WaitOne();

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(Product[product] + " bought");

Console.ResetColor();

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

int i = product;

int Counter = 0;

Warehouse.TryTake(out i);

Console.WriteLine("Warehouse: ");

foreach (var productt in Warehouse)

{

Console.WriteLine(Product[productt]);

Counter++;

}

if (Counter == 0)

{

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("Nothing");

Console.ResetColor();

}

else

Counter = 0;

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

mutex.ReleaseMutex();

break;

}

}

});

}

Thread.Sleep(5000);

}

public static async void AsyncDemonstration()

{

await Task.Run(Continuation);

}

}

}

# Сериализация

using System;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

using System.Xml.Serialization;

using System.Collections.Generic;

using System.Text.Json;

//using System.Runtime.Serialization.Formatters.Soap;

using System.Runtime.Serialization.Json;

using System.Runtime.Serialization;

namespace Serialization

{

class Program

{

static void Main(string[] args)

{

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

Users.Add(new User(){ Login = "user1", Name = "Bob", Password = "123"});

Users.Add(new User(){ Login = "user2", Name = "Nick", Password = "343"});

Users.Add(new User(){ Login = "user3", Name = "Ivan", Password = "562"});

Users.Add(new User(){ Login = "user4", Name = "Josh", Password = "294"});

Serialization.XML(Users);

}

[Serializable]

public class User

{

public string Name { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public User(){}

}

internal static class Serialization

{

public static void XML(List<User> Users)

{

XmlSerializer xml = new XmlSerializer(typeof(User));

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

{

xml.Serialize(fs, Users);

}

}

public static void JSON(List<User> Users)

{

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

{

fs.WriteLine(JsonSerializer.Serialize(Users));

}

}

public static void Binary(List<User> Users)

{

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

{

BinaryFormatter bin = new BinaryFormatter();

bin.Serialize(fs, Users);

}

}

//public static void SOAP(List<User> Users)

//{

// SoapFormatter soap = new SoapFormatter();

// using(FileStream fs = new FileStream("soap.soap", FileMode.Create))

// {

// soap.Serialize(fs, Users);

// }

//}

public static void JSONContracts(List<User> Users)

{

DataContractJsonSerializer json = new DataContractJsonSerializer(typeof(List<User>));

using (FileStream fs = new FileStream("json1.json", FileMode.Create))

{

json.WriteObject(fs, Users);

}

}

public static void XMLContracts(List<User> Users)

{

DataContractSerializer xml = new DataContractSerializer(typeof(List<User>));

using(FileStream fs = new FileStream("xml1.xml", FileMode.Create))

{

xml.WriteObject(fs, Users);

}

}

}

}

}

# События

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();

}

}

}

}

# Атрибуты

using System;

namespace ConsoleApplication1

{

// Создаем атрибут

[AttributeUsage(AttributeTargets.Method | AttributeTargets.Field)]

public sealed class UInfoAttribute : System.Attribute

{

public string Desc;

public UInfoAttribute() { }

public UInfoAttribute(string str)

{

Desc = str;

}

}

class Program

{

[UInfo(Desc = "Главный метод программы")]

static void Main()

{

Type t = typeof(UInfoAttribute);

object[] obj = t.GetCustomAttributes(false);

foreach (object o in obj)

Console.WriteLine(o);

Console.ReadLine();

}

}

}