**Лабораторная 2. Основы .NET Массивы. Кортежи. Строки.**

using System;

namespace **Array**

{

class Array

{

static void Main(string[] args)

{

// двумерный массив и вывод в виде матрицы

Console.WriteLine("Двумерный массив и вывод в виде матрицы");

int[,] mas = { { 1, 2 }, { 2, 1 } };

int rows = mas.GetUpperBound(0) + 1; //Получает Индекс последнего элемента заданного измерения в массиве.

int columns = mas.Length / rows;

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

{

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

{

Console.Write($"{mas[ii, ij]} ");

}

Console.WriteLine();

}

//одномерный массив строк

Console.WriteLine("Одномерный массив строк");

string[] items = { "ноутбук", "мышь", "система", "экран", "клавиатура" };

for (int ij = 0; ij < items.Length; ++ij)

{

Console.WriteLine(items[ij] + " ");

}

Console.WriteLine($"Длина массива: { items.Length}");

Console.Write("Введите позицию элемента массива, который вы хотите заменить: ");

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

Console.Write("Введите значение элемента массива: ");

string value = Console.ReadLine();

items[position] = value;

for (int ij = 0; ij < items.Length; ++ij)

{

Console.Write(items[ij] + " ");

}

Console.WriteLine();

//ступенчатый массив вещественных чисел с 3-мя строками

/\*Если требуется очень длинный двумерный массив,

\* который заполняется не полностью,

\* т.е. такой массив, в котором используются не все,

\* а лишь отдельные его элементы. \*/

Console.WriteLine($"\nСтупенчатый массив вещественных чисел с 3-мя строками");

int[][] arr = new int[3][];

arr[0] = new int[2];

arr[1] = new int[3];

arr[2] = new int[4];

Console.WriteLine("Введите 9 элементов массива через пробел: ");

string[] arrayNumber = Console.ReadLine().Split(' ');

int j = 0;

do

{

Console.WriteLine();

for (int li = 0; li < 2; ++li, j++)

{

arr[0][li] = int.Parse(arrayNumber[j]);

Console.Write(arr[0][li] + " ");

}

Console.WriteLine();

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

{

arr[1][li] = int.Parse(arrayNumber[j]);

Console.Write(arr[1][li] + " ");

}

Console.WriteLine();

for (int li = 0; li < 4; ++li, j++)

{

arr[2][li] = int.Parse(arrayNumber[j]);

Console.Write(arr[2][li] + " ");

}

} while (j < arrayNumber.Length);

// неявно типизированные переменные для хранения массива и строки

var array3 = new[] { "1", "2", "3", "4", "5" };

var string3 = "Abcdefg";

Console.WriteLine("\nНеявно типизированные переменные для хранения массива и строки:" + array3 + " и " + string3);

}

}

}

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

/\*Так, если требуется указать, что выражение будет проверяться на переполнение,

\* следует использовать ключевое слово checked,

\* а если требуется проигнорировать переполнение — ключевое слово unchecked.

\* В последнем случае результат усекается, чтобы не выйти за пределы диапазона представления чисел для целевого типа выражения. \*/

namespace **CheckedUnchecked**

{

class CheckedUnchecked

{

static void Main(string[] args)

{

int Fun1()

{

checked

{

int x = int.MaxValue;

try

{

return x + 1;

}

catch (OverflowException)

{

Console.WriteLine("Переполнение");

return x;

}

}

}

int Fun2()

{

unchecked

{

int x = int.MaxValue;

try

{

return x + 1;

}

catch (OverflowException)

{

Console.WriteLine("Переполнение");

return x;

}

}

}

Console.WriteLine(Fun1());

Console.WriteLine(Fun2());

}

}

}

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

using System.Linq; //LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных.

/\*Локальные функции — это закрытые методы типа,

\* вложенные в другой член. Они могут быть вызваны только из содержащего их члена.\*/

namespace **LocalFunction**

{

class LocalFunction

{

static void Main(string[] args)

{

(int, int, int, char) localFun(int[] mass, string st)

{

int max = mass.Max();

int min = mass.Min();

int sum = mass.Sum();

char sim = st.First();

return (max, min, sum, sim);

}

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

string st = "Hello, World";

Console.WriteLine(localFun(mass, st));

}

}

}

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

using System.Text;

/\*Шаблонными литералами называются строковые литералы,

\* допускающие использование выражений внутри,

\* обозначаемых знаком $ и фигурными скобками (${выражение}).

\* Позволяют использовать многострочные литералы и строковую интерполяцию.

\*/

namespace **Strings**

{

class Strings

{

static void Main(string[] args)

{

Console.WriteLine("Введите номер задания (1-4), по которому нужно вывести результат:");

string selection = Console.ReadLine();

switch (selection)

{

case "1":

First();

break;

case "2":

Second();

break;

case "3":

Third();

break;

case "4":

Fourth();

break;

default:

Console.WriteLine("Вы ввели неправильную цифру!");

Environment.Exit(0);

break;

}

}

static void First()

{

char[] a1 = { 'w', 'o', 'r', 'd' };

string string1 = new string(a1);

Console.WriteLine(string1);

string string2 = new string('х', 20);

Console.WriteLine(string2);

string string3 = null;

Console.WriteLine(string3);

Console.WriteLine("Oh, hello");

Console.WriteLine("Привет всем \"Вам\"");

Console.WriteLine("Привет \nи мне\n");

}

static void Second()

{

// объединение //

string s1 = "Конкан";

string s2 = "тенация";

string s3 = s1 + s2;

Console.WriteLine(s1 + s2);

Console.WriteLine(String.Concat(s3));//Метод Concat является статическим методом класса String

string s5 = "Чисти";

string s6 = "зубы";

string s7 = "каждый";

string s8 = "день";

string s9 = "обязательно!";

string[] values = new string[] { s5, s6, s7, s8, s9 };

String s10 = String.Join(" ", values);

Console.WriteLine(String.Join(" ", values) + " " + s10);

Console.WriteLine($"{s1}{s2}");

int x = 8;

int y = 7;

string result = $"{x} + {y} = {x + y}";

Console.WriteLine(result); // 8 + 7 = 15

//Интерполяция - форматирование строки - способ соединения строк через вставку значений переменных в строку-шаблон с помощью фигурных скобок.

// копирование строк //

// Clone - возвращает ссылку на данный экземпляр класса String.

string p = s1;

char[] charArr = new char[] { 'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd' };

string s11 = "string";

s11.CopyTo(1, charArr, 0, 5);

Console.WriteLine(charArr);

//string s4 = "Hello";

//string s54 = string.Copy(s4);

//Console.WriteLine(s54);

// выделение подстроки

string newString = "Привет тебе и тебе";

string subNewString = "тебе";

int indexOfSub = newString.IndexOf(subNewString);

int indexOfSub1 = newString.LastIndexOf(subNewString);

Console.WriteLine("Первый индекс: " + indexOfSub + ", последний индекс: " + indexOfSub1);

// разделение строки на слова//

string s = "You win. You lose.";

string[] subs = s.Split(' ');

foreach (var sub in subs)

{

Console.WriteLine($"Substring: {sub}");

}

// вставка подстроки //

string text = "Хороший день";

string subString = "сегодня ";

text = text.Insert(8, subString);

Console.WriteLine(text);

// удаление заданной подстроки //

// индекс последнего символа

int ind = text.Length - 3;

// вырезаем последний символ

text = text.Remove(ind);

Console.WriteLine(text);

// вырезаем первые два символа

text = text.Remove(0, 2);

Console.WriteLine(text);

string phrase = "look at me now";

Console.WriteLine("До: " + phrase);

phrase = phrase.Replace(" now", "");

Console.WriteLine("После: " + phrase);

}

static void Third()

{

string emptyString = "";

string emptyString1 = string.Empty;

string nullString = null;

string notAnEmpty = "Не путая строка";

string[] array = { emptyString, nullString };

Console.WriteLine(String.IsNullOrEmpty(emptyString));

Console.WriteLine(String.IsNullOrEmpty(emptyString1));

Console.WriteLine(String.IsNullOrEmpty(nullString));

Console.WriteLine(String.IsNullOrEmpty(notAnEmpty));

string con = emptyString + nullString; //сцепление

Console.WriteLine(con);

Console.WriteLine("Equals: " + (emptyString.Equals(nullString)));

Console.WriteLine("CompareTo: " + emptyString.CompareTo(nullString));

Console.WriteLine("String.Concat: " + String.Concat(nullString, emptyString));

Console.WriteLine("Interpolation: " + $"{nullString}{emptyString}");

}

static void Fourth()

{

//StringBuilder - класс - Предоставляет изменяемую строку символов. Этот класс не наследуется.

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

Console.WriteLine($"Удаление c 4 позиций 2-ух символов: {sb.Remove(4, 2)}");

sb = sb.Insert(0, "!!!");

sb = sb.Insert(13, "!!!");

Console.WriteLine($"Добавление новых символов в начало и конец строки: {sb}");

}

}

}

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

namespace **Tuple – кортеж**

{

class Tuple

{

static void Main(string[] args)

{

(int, string, char, string, ulong) tuple1 = (19, "Василиса", 'ж', "Кашперко", 2001);

Console.Write(tuple1.Item1 + " " + tuple1.Item2 + " " + tuple1.Item4);

(var ar, var br) = ("2001", 2002);

Console.WriteLine($"\n{ar} {br}" + "\n");

var First = (ar, br);

var Second = (ar, br);

Console.WriteLine(First == Second);

}

}

}

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

class **Types**

{

static void Main(string[] args)

{

////////// ЗАДАНИЕ 1 - инициализация переменных //////////

Console.WriteLine($"size of int = {sizeof(int)}");

Console.WriteLine($"size of bool = {sizeof(bool)}");

bool valueBool = true;

Console.WriteLine(valueBool ? "Checked" : "Not checked" + " " + valueBool.GetType());

Console.WriteLine(true ? "Checked" + "\n" : "Not checked"+"\n");

sbyte valueSByte = 110; //От -128 до 127. Знаковое 8-битное целое число

Console.WriteLine(valueSByte + " " + valueSByte.GetType()+"\n");

byte valueByte = 255; // От 0 до 255. Беззнаковое 8-битное целое число

Console.WriteLine(valueByte + " " + valueByte.GetType());

Console.WriteLine(valueByte + " " + Convert.ToInt32(valueByte).GetType());

Console.WriteLine(valueByte + " " + valueByte.GetType() + "\n");

short valueShort = 31700;//От -32 768 до 32 767. Знаковое 16-битное целое число

Console.WriteLine(valueShort + " "+ valueShort.GetType()+"\n");

ushort valueUShort = 65535;//От 0 до 65 535. Беззнаковое 16-битное целое число

Console.WriteLine(valueUShort + " " + valueUShort.GetType()+"\n");

int valueInt = -2147483647;//От - 2 147 483 648 до 2 147 483 647. 32 - битное целое со знаком

Console.WriteLine(valueInt + " " + valueInt.GetType() + "\n");

uint valueUInt = 0;//От 0 до 4 294 967 295. Беззнаковое 32-битное целое число

Console.WriteLine(valueUInt + " " + valueUInt.GetType() + "\n");

long valueLong = -930492842343423; //От -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807. 64-битное целое число со знаком

Console.WriteLine(valueLong + " " + valueLong.GetType() + "\n");

ulong valueUlong = 2322222222222222222; //От 0 до 18 446 744 073 709 551 615. Беззнаковое 64-битное целое число

Console.WriteLine(valueUlong + " " + valueUlong.GetType() + "\n");

char valueChar = 'A'; // хранит одиночный символ в кодировке Unicode и занимает 2 байта.

char valueChar1 = '\u0420'; // \u - юникод, четырехсимвольное шестнадцатеричное представление символьного кода; \x - шестнадцатиричное значение

Console.WriteLine(valueChar + " " + valueChar.GetType() + " "+ valueChar1 + " " + valueChar1.GetType() + "\n");

double valueDouble = 12.3;

Console.WriteLine(valueDouble + " " + valueDouble.GetType() + "\n");

decimal valueDecimal = 0.251m;

Console.WriteLine(valueDecimal + " " + valueDecimal.GetType() + "\n");

float valueFloat = 3.1456789234f;

Console.WriteLine(valueFloat + " " + valueFloat.GetType() + "\n");

/\*

nint valueNInt = 8; //Зависит от платформы. 32 - битное или 64 - битное целое число со знаком

Console.WriteLine($"size of nint = {sizeof(nint)}");

Console.WriteLine($"size of nuint = {sizeof(nuint)}");

// Использование версии языка preview.

\*/

object valueObject = "Hello";

object valueObject1 = 3.14;

Console.WriteLine($"Объект 1: {valueObject}\nОбъект 2: {valueObject1}");

Console.WriteLine(valueObject.GetType() + " " + valueObject1.GetType() + "\n");

var valueVar = 93;

Console.WriteLine(valueVar + " " + valueVar.GetType());

valueVar = 100;

//valueVar = 100.9;

Console.WriteLine(valueVar + " " + valueVar.GetType());

//valueVar = "Hi";

Console.WriteLine(valueVar + " " + Convert.ToSingle(valueVar).GetType());

Console.WriteLine(valueVar + " " + valueVar.GetType() + "\n");

dynamic valueDynamic = "Hi";

Console.WriteLine(valueDynamic + " " + valueDynamic.GetType());

valueDynamic = 93;

Console.WriteLine(valueDynamic + " " + valueDynamic.GetType() + "\n");

dynamic dyn = 1;

object obj = 1;

dyn = dyn + 3;

Console.WriteLine(dyn+"\n");

//obj = obj + 3;

////////// ЗАДАНИЕ 2 - явное и неявное приведения //////////

byte a = 4;

int b = a + 70;

//byte a = 4;

//byte b = a + 70; // ошибка

byte c = 4;

byte d = (byte)(c + 70);

ushort e = 4;

byte f = (byte)e; // для сужающего преобразования

int g = 4;

int h = 6;

byte i = (byte)(g + h);

var j = (dynamic)"Привет";

bool k = (bool)true;

double l = double.Parse("23,56");

float m = 392.222f;

dynamic o = 9.333333m;

int r = (int)1.34;

var p = "Строчка";

double q = 23.4;

int ar = 1;

long s = 1;

ushort t = 42;

Console.WriteLine($"a = {a}, b = {b}, c = {c}, d = {d}," +

$"e = {e}, f = {f}, g = {g}, h = {h}, " +

$"i = {i}, j = {j},\nk = {k}, l = {l}, " +

$"m = {m}, o = {o}, p = {p}, q = {q}, " +

$"ar = {ar}, r = {r},\ns = {s}, t = {t}\n");

Console.WriteLine(valueVar + " " + valueVar.GetType() + "\n");

Console.WriteLine(valueVar + " " + Convert.ToSingle(valueVar).GetType());

int n = Convert.ToInt32("46");

string dateString = "05/01/1996";

Convert.ToDateTime(dateString);

Console.WriteLine($"n = {n} и dateString = {dateString} и при конвертации " + Convert.ToDateTime(dateString));

////////// ЗАДАНИЕ 3 - упаковка/распаковка значимых типов //////////

int price = 39;

Object obj1 = price;

int price2 = (int)obj1;

double eco = 2.93;

Object obj2 = eco;

// eco = (Int32)eco; пример распаковки в int - округление в меньшую сторону

Console.WriteLine(" ");

Console.WriteLine(eco);

Console.WriteLine(eco.GetType());

////////// Задание 4 - неявно типизированная переменная ///////////

var notADataType = new float[10];

// notADataType = 4; // ошибка, т.к. сторгая типизация

Console.WriteLine(notADataType.GetType());

var notADataType2 = 32;

var notADataType3 = 2;

Console.WriteLine(notADataType2+notADataType3);

//var notADataType4 = null; // null - такое значение принимает ссылочный тип или nullable, но какая переменная?)) не понятно!

var notADataType5 = "String";

notADataType5 = null;

// или var notADataType = (string)null;

Console.WriteLine(notADataType5);

////////// Задание 5 - пример работы с nullable ///////////

int? n1 = null;

bool? n2 = null;

Nullable<bool> n3 = true;

n3 = null;

Console.WriteLine(n1 + " - nullable int;" + n2 + " - nullable bool;" + n3 + " - nullable bool после инициализации;");

int x1 = 4;

int? x2 = x1;

x2 = null;

Console.WriteLine(x2 + " - nullable неявное преобразование к null");

////////// Задание 6 - пример работы с nullable ///////////

var hello = 2;

Console.WriteLine(hello);

//hello = "Привет"; - исключение

}

}

**Лабораторная 3. Проектирование типов. Классы.**

using System;

/\*

Создать класс Customer: id, Фамилия, Имя, Отчество, Адрес, Номер кредитной карточки, баланс. Свойства и конструкторы должны обеспечивать проверку корректности. Добавить методы зачисления и списания сумм на счет.

Создать массив объектов. Вывести:

a) список покупателей в алфавитном порядке;

b) список покупателей, у которых номер кредитной карточки

находится в заданном интервале.

\*/

namespace Lab3

{

public partial class Customer //Частичные классы (partial classes) – конструкция, которая позволяет определить один класс в нескольких файлах.

{

public static string about = "Этот класс о клиентах.";

private static int count; //Ключевое слово static - объявление константы или типа неявно является членом static. На член static невозможно ссылаться через экземпляр, а можно только через имя типа.

private readonly Guid id;

private string surname;

private string name;

private string patronymic;

private string address;

private ulong numOfCard;

private ulong balance;

// Свойства

public Guid Id { get => id; }

public string Surname { get => surname; set { surname = value; } }

public string Name { get => name; set { name = value; } }

public string Patronymic { get => patronymic; set { patronymic = value; } }

public string Address { get => address; set { address = value; } }

public ulong NumOfCard

{

get { return numOfCard; }

set

{

if (value > 0000000000000001 && value <= 9999999999999999)

numOfCard = value;

else

Console.WriteLine("Неверный номер карты!");

}

}

public ulong Balance { get => balance; set { balance = value; } }

//Конструкторы

// с параметрами

public Customer(Guid id, string surname, string name, string patronymic, string address, ulong numOfCard, ulong balance)

{

if (numOfCard > 0000000000000001 && numOfCard <= 9999999999999999)

{

this.id = id;

this.surname = surname;

this.name = name;

this.patronymic = patronymic;

this.address = address;

this.numOfCard = numOfCard;

this.balance = balance;

Customer.count++;

}

else

throw new Exception("Некорректный ввод данных! Проверьте номер карты или id клиента.");

}

// без параметров

public Customer()

{

surname = "Котов";

name = "Николай";

patronymic = "Алексеевич";

address = "г.Гродно, ул. Партизанская, д. 62, кв. 1";

numOfCard = 1111111111111111;

balance = 88727319918;

}

// с параметрами по умолчанию

public Customer(Guid id, string patronymic, ulong numOfCard, ulong balance, string surname = "Мышович", string name = "Игорь")

{

this.id = id;

this.surname = surname;

this.name = name;

this.patronymic = patronymic;

this.numOfCard = numOfCard;

this.balance = balance;

}

// статический

//Статический конструктор используется для инициализации

//любых статических данных или для выполнения определенного действия,

//которое требуется выполнить только один раз.

//static Customer()

//{

// Console.WriteLine("Ваши данные:");

//}

// закрытый

//Закрытые конструкторы используются, чтобы не допустить

//создания экземпляров класса при отсутствии полей или методов экземпляра

private Customer(Guid id, string surname)

{

this.id = id;

this.surname = surname;

}

// предотвращает создание объекта класса из-вне. Используется в синглтонах и фабриках.

public override string ToString() // переопределение метода ToString

{

return $"ID: {this.id}\n" +

$"Фамилия: {this.surname}\n" +

$"Имя: {this.name}\n" +

$"Отчество: {this.patronymic}\n" +

$"Адрес: {this.address}\n" +

$"Номер карты: {this.numOfCard}\n" +

$"Баланс: {this.balance} рублей\n";

}

// Метод вывода фамилии клиента

public void Print(ref Customer customer, out string printSurname)

{

printSurname = customer.surname;

Console.WriteLine(printSurname);

}

// Метод вывода информации о классе

public static string GetInfoAboutClass()

{

return about;

}

public void PrintIt()

{

Console.WriteLine(this.ToString());

}

}

}

using System;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

try

{

Customer customer1 = new Customer(Guid.NewGuid(), "Игоренко", "Николай", "Васильевич", "г.Могилёв, ул. Крестьянская, д. 21, кв. 52", 7340930482768764, 36229);

Customer customer2 = new Customer();

Customer[] customers = new Customer[7];

customers[0] = new Customer(Guid.NewGuid(), "Смирнов", "Павел", "Валентинович", "г.Гомель, ул. Советская, д. 25а, кв. 24", 2343253487549364, 26439);

customers[1] = new Customer(Guid.NewGuid(), "Павлов", "Игорь", "Сергеевич", "г.Минск, ул. Белорусская, д. 21, кв. 36", 6284275630582759, 14256);

customers[2] = new Customer(Guid.NewGuid(), "Миронов", "Владислав", "Николаевич", "г.Брест, ул. Клермон Ферран, д. 2, кв. 1", 1038472837502758, 1223145);

customers[3] = new Customer(Guid.NewGuid(), "Тимонович", "Прохор", "Викторович", "г.Витебск, ул. Чижова, д. 115, кв. 95", 4839164727361847, 6432);

customers[4] = new Customer(Guid.NewGuid(), "Даниленко", "Кирилл", "Александрович", "г.Новополоцк, ул. Владимирская, д. 52б, кв. 83", 2847293827460192, 162423);

customers[5] = customer1;

customers[6] = customer2;

Array.Sort(customers, (x, y) => String.Compare(x.Surname, y.Surname));

Console.WriteLine("\t\t\tСписок клиентов в алфавитном порядке:\n");

foreach (Customer customer in customers)

{

Console.WriteLine(customer.ToString());

}

Console.ReadLine();

Console.WriteLine("\t\tСписок клиентов, у которых номер кредитной карточки");

Console.WriteLine("\t\t\tнаходится в заданном интервале:\n");

foreach (var customer in customers)

{

if ( 3000000000000000 < customer.NumOfCard && customer.NumOfCard < 9000000000000000 )

{

customer.PrintIt();

}

}

Console.ReadLine();

// Анонимный тип

//Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса.

var anonimuscustomer = new { surname = "BSTU", name = "Kirov", balance = 1033 };

Console.WriteLine(anonimuscustomer.ToString());

Console.WriteLine(anonimuscustomer.GetType());

}

catch (Exception excpt)

{

Console.WriteLine(excpt);

}

}

}

}

**Лабораторная 4. Перегрузка. Методы расширения.**

using System;

using System.Collections.Generic; // простанство имен, содержащее реализацию классов типа список, массив, очередь

using System.Linq;

// Коллекция, содержащая только отличающиеся элементы, называется множеством (set).

// Класс List<T> из пространства имен System.Collections.Generic представляет простейший список однотипных объектов.

namespace Lab4

{

/// <summary>

/// Множество

/// </summary>

/// <typeparam name="T"> Тип данных, хранимых во множестве. </typeparam>

public partial class Set

{

public List<int> mySet;

public Set(List<int> localSet)

{

this.mySet = localSet;

}

// Удаление элемента из множества

public static Set operator -(Set set1, int item1)

{

set1.mySet.Remove(item1); // удаление из mySet (хранилища)

return set1;

}

// Пересечение множеств

public static Set operator \*(Set set1, Set set2)

{

var concSet = new Set(new List<int>());

foreach (var s in set1.mySet)

{

if (set2.mySet.Contains(s))

{

concSet.mySet.Add(s);

}

}

return concSet;

}

// Сравнение множеств

public static bool operator <(Set set1, Set set2)

{

int amount1 = set1.mySet.Count;

int amount2 = set2.mySet.Count;

if (amount1 <= amount2)

{

Console.WriteLine("1 множество меньше 2");

return false;

}

else

{

Console.WriteLine("1 множество больше 2");

return true;

}

}

// Проверка на подмножество

public static bool operator >(Set set1, Set set2)

{ // 1-ый параметр - подмножесто, 2-ой - множество

int c = set1.mySet.Count;

int counter = 0;

foreach (var s in set1.mySet)

{

if (set2.mySet.Contains(s))

{

counter++;

}

}

if (counter == c)

{

return true;

}

else

{

return false;

}

}

// Добавление элемента в множество

public static Set operator &(Set set1, int q)

{

if (set1.mySet.Contains(q))

{

Console.WriteLine("Такой элемент уже есть в множестве.");

}

else

{

set1.mySet.Add(q);

}

return set1;

}

public void Print()

{

this.mySet.ForEach(i => Console.Write(i + " "));

Console.WriteLine();

}

public static void SetTest()

{

var set1 = new Set(new List<int>() { 0, 1, 2, 3, 5, 6, 7, 9 });

var set2 = new Set(new List<int>() { 1, 2, 3, 5, 7 });

set1.Print();

set2.Print();

Console.WriteLine("Удаляем из первого множества 5");

var test = set1 - 5;

test.Print();

Console.WriteLine("Добавляем в первое множество 10");

test = set1 & 10;

test.Print();

Console.WriteLine("Пересечение множеств: ");

set1.Print();

set2.Print();

Console.WriteLine("Результат");

test = set2 \* set1;

test.Print();

Console.WriteLine("Сравниваем два множества");

Console.WriteLine(set1 < set2);

Console.WriteLine("Включает ли первое множество в себе второе");

set1.Print();

set2.Print();

Console.WriteLine(set2 > set1);

test = set2 - 5;

set1.Print();

set2.Print();

Console.WriteLine(set2 > set1);

}

}

public static class StatisticOperation

{

public static string AddPoint(this string str)

{

str += '.';

return str;

}

public static Set RemoveZero(this Set set1)

{

while (set1.mySet.Remove(0))

{

}

return set1;

}

public static int Sum(Set set1, Set set2)

{

return set1.mySet.Count + set2.mySet.Count;

}

public static int Diff(Set set1)

{

return set1.mySet.Max() - set1.mySet.Min();

}

public static int Length(Set set1)

{

return set1.mySet.Count;

}

}

}

using System;

namespace Лабораторная\_4

{

public partial class Set

{

public class Owner

{

public Guid ID;

public string Name;

public string Organization;

public Owner()

{

ID = Guid.NewGuid();

Name = "Vasilisa";

Organization = "BSTU";

}

}

public Owner myOwner = new Owner();

public static class Date

{

public static DateTime date = new DateTime(2021, 10, 24);

}

}

}

using System;

namespace Lab4

{

class Program

{

static void Main(string[] args)

{

Set.SetTest();

string str = "Это новая строка";

Console.WriteLine(str.AddPoint());

}

}

}

**Лабораторная 5. Наследование. Полиморфизм. Абстрактные интерфейсы.**

using System;

namespace Лабораторная\_5

{

interface IRemove

{

void Recycle();

}

class Bouquet : IRemove

{

public string BouquetName { set; get; }

public Paper BouquetPaper { set; get; }

public Flower[] Flowers = new Flower[3];

public Bouquet(string bouquetName, Paper myPaper, Rose myRose, Glasiolus myGlasiolus, Cactus myCactus)

{

BouquetName = bouquetName;

BouquetPaper = myPaper;

Flowers[0] = myRose;

Flowers[1] = myGlasiolus;

Flowers[2] = myCactus;

}

public override string ToString()

{

return $"{GetType()} {BouquetName} {BouquetPaper} {Flowers[0]} {Flowers[1]} {Flowers[2]}";

}

public void Recycle()

{

Console.WriteLine($"Букет {BouquetName} отправлен в переработку. ");

}

}

}

using System;

namespace Лабораторная\_5

{

class Bush : Plant, IRemove

{

public string GrowLocation

{

get; set;

}

public Bush(string name, string growLocation, string lifeForm = "Кустарник") : base(name, lifeForm)

{

GrowLocation = growLocation;

}

void IRemove.Recycle()

{

Console.WriteLine($"Куст {Name} переработали! \*Интерфейс\*");

}

public override void Recycle()

{

Console.WriteLine($"Куст {Name} переработали! \*Абстрактный класс\*");

}

virtual public void Uproot()

{

Console.WriteLine($"Куст {Name} был выкорчеван.");

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm} {GrowLocation}";

}

}

}

namespace Лабораторная\_5

{

class Cactus : Flower

{

public bool HasSingleFlower

{

get; set;

}

public Cactus(string name, string growLocation, string color, int diameter) : base(name, growLocation, color, diameter)

{

HasSingleFlower = true;

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm} {GrowLocation} {Color} {Diameter} {HasSingleFlower}";

}

}

}

using System;

namespace Лабораторная\_5

{

class Flower : Bush, IRemove

{

public string Color

{

get; set;

}

public int Diameter

{

get; set;

}

public Flower(string name, string growLocation, string color, int diameter) : base(name, growLocation)

{

Color = color;

Diameter = diameter;

}

public override void Uproot()

{

Console.WriteLine($"Цветок {Name} срезан для букета.");

}

public override void Recycle()

{

Console.WriteLine($"Цветок {Name} выбросили. Жаль. Получился бы отличный букет :(");

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm} {GrowLocation} {Color} {Diameter}";

}

}

}

using System;

namespace Лабораторная\_5

{

class Glasiolus : Flower

{

public int FlowersOnStem

{

get; set;

}

public Glasiolus(string name, string growLocation, string color, int diameter, int flowersOnStem) : base(name, growLocation, color, diameter)

{

FlowersOnStem = flowersOnStem;

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm} {GrowLocation} {Color} {Diameter} {FlowersOnStem}";

}

}

}

using System;

namespace Лабораторная\_5

{

sealed class Paper

{

public string Color { set; get; }

public bool HasUniqueStyle { set; get; }

public bool IsTransparent { set; get; }

public Paper(string color)

{

Color = color;

HasUniqueStyle = true;

IsTransparent = false;

}

public override string ToString()

{

return $"{GetType()} {Color} {HasUniqueStyle} {IsTransparent}";

}

}

}

using System;

namespace Лабораторная\_5

{

abstract class Plant

{

public abstract void Recycle();

public string Name

{

get; set;

}

public string LifeForm

{

get; set;

}

public Plant(string name, string lifeForm)

{

Name = name;

LifeForm = lifeForm;

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm}";

}

}

}

using System;

namespace Лабораторная\_5

{

class Rose : Flower

{

public bool HasThorns

{

get; set;

}

public int ThornLenth

{

get; set;

}

public Rose(string name, string growLocation, string color, int diameter, int thornLength) : base(name, growLocation, color, diameter)

{

ThornLenth = thornLength;

HasThorns = true;

}

public override void Recycle()

{

Console.WriteLine("Пожалуй розы лучше подарить, чем выбрасывать!");

}

public override string ToString()

{

return $"{this.GetType()} {Name} {GrowLocation} {Color} {Diameter} {HasThorns} {ThornLenth}";

}

public override int GetHashCode()

{

return Name.GetHashCode() + ThornLenth.GetHashCode();

}

public override bool Equals(object obj)

{

if (obj != null)

{

Rose m = obj as Rose; // возвращает null если объект нельзя привести к типу Roze

if (m as Rose == null)

return false;

return m.Name == this.Name && m.ThornLenth == this.ThornLenth;

}

else

{

return false;

}

}

}

}

using System;

// 1 - Flower

// 2 - Plant, Bush

// 3 - Paper

// 4 - Bush (интерфейс - Bouquet, абстрактный класс - Plant)

// 5 - Program (as, is)

// 6 - Во всех

// 7 - Program

namespace Лабораторная\_5

{

class Printer

{

public void IAmPrinting(Plant obj)

{

Console.WriteLine(obj.ToString());

}

}

class Program

{

static void Main(string[] args)

{

Bush myBush = new Bush("Шиповник", "Лес");

Flower myFlower = new Flower("Тюльпан", "Поле", "Желтый", 6);

Rose myRose = new Rose("Роза", "Поле", "Красный", 5, 2);

Glasiolus myGlad = new Glasiolus("Гладиолус", "Лес", "Белый", 3, 4);

Cactus myCactus = new Cactus("Алоэ", "Пустыня", "Зеленый", 3);

Paper myPaper = new Paper("Розовый");

Bouquet myBouquet = new Bouquet("Дар весны", myPaper, myRose, myGlad, myCactus);

Console.WriteLine($"{myBush} \n {myFlower} \n {myRose} \n {myGlad} \n {myCactus} \n {myPaper} \n {myBouquet}");

Plant myPlant = new Bush("Малина", "Лес"); // объект Plant создать нельзя, можно хранить объекты производных классов

//Преобразование Plant в Bush

Console.WriteLine(myBush.Name);

Console.WriteLine(myPlant.Name + " " + myPlant.LifeForm); // не можем вывести myPlant.growLocation, нужно преобразовать к Bush

myBush = myPlant as Bush;

if (myBush == null)

{

Console.WriteLine("Преобразование НЕ удалось");

}

else

{

Console.WriteLine(myBush.Name + " " + myBush.GrowLocation);

}

//Преобразование Rose в Flower

Console.WriteLine(myFlower.Name);

if (myRose is Flower)

{

myFlower = (Flower)myRose;

Console.WriteLine(myFlower.Name); // Roze преобразован во Flower, получить доступ к .ThornsLength невозможно.

}

else

{

Console.WriteLine("Преобразование не допустимо");

}

//Ссылка на Bush типа IRemove

IRemove theBush = new Bush("Смородина", "Лес");

//Вызовется метод интерфейса т. к. ссылка типа IRemove

theBush.Recycle();

Plant newBush;

//Преобразование IRemove в Plant

newBush = theBush as Plant;

if (newBush == null)

{

Console.WriteLine("Преобразование НЕ удалось");

}

else

{

Console.WriteLine(newBush.Name); //вывести поле growLocation не получится.

//Вызовется метод Recycle() от Plant, т. к. ссылка типа Plant

newBush.Recycle();

}

Printer myPrinter = new Printer();

dynamic[] myArray = new dynamic[] { myPlant, myBush, myFlower, myRose, myGlad, myCactus };

foreach (var s in myArray)

{

myPrinter.IAmPrinting(s);

}

}

}

}

**Лабораторная 6. Структуры. Перечисления. Контейнеры. Контроллеры.**

using System;

public static class Controller

{

public static void SortByName(Bouquet bouquet)

{

bouquet.Flowers.Sort();

}

public static void FindByColor(Bouquet bouquet, string color)

{

foreach (var s in bouquet.Flowers)

{

if (s.Color == color)

{

Console.WriteLine(s);

}

}

}

}

namespace Лабораторная\_6

{

class Printer

{

public void IAmPrinting(Plant obj)

{

Console.WriteLine(obj.ToString());

}

}

interface IRemove

{

void Recycle();

}

class Program

{

static void Main(string[] args)

{

Bush myBush = new Bush("Шиповник", "Равнина");

Flower myFlower = new Flower("Тюльпан", "Лес", "Желтый", 6);

Rose myRoze = new Rose("Роза голандская", "Равнина", "Красный", 5, 2);

Glasiolus myGlad = new Glasiolus("Гладиолус", "Лес", "Белый", 3, 4, Season.Autumn);

Cactus myCactus = new Cactus("Алоэ", "Пустыня", "Салатовый", 3);

Paper myPaper = new Paper("Розовый");

Plant myPlant = new Bush("Малина", "Лес"); // объект Plant создать нельзя, может хранить объекты производных классов

//lab6

Console.WriteLine($"{Season.Winter}, {Season.Spring}, {Season.Summer}, {Season.Autumn}");

Console.WriteLine($"{(int)Season.Winter}, {(int)Season.Spring}, {(int)Season.Summer}, {(int)Season.Autumn}");

Buyer person1;

//person1.ShowInfo(); Вызвать метод объекта нельзя, нужно проинициализировать поля.

person1.Name = "Lisi";

person1.Age = 19;

person1.HasDiscount = true;

person1.ShowInfo();

Buyer person2 = new Buyer();

person2.Name = "Jack";

person2.Age = 29;

person2.HasDiscount = false;

person2.ShowInfo();

Buyer person3 = new Buyer("Tom", 22, true);

person3.ShowInfo();

Bouquet myBouquet = new Bouquet("Букет Рассвет", myPaper, myRoze, myGlad, myCactus);

myBouquet.GetInfo();

Flower myTulpan = new Flower("Тюльпан", "Лес", "Желтый", 6);

myBouquet.AddFlower(myTulpan);

myBouquet.GetInfo();

Controller.SortByName(myBouquet);

myBouquet.GetInfo();

Controller.FindByColor(myBouquet, "Желтый");

}

}

}

namespace Лабораторная\_6

{

// 6

enum Season

{

Winter = 1,

Spring,

Summer,

Autumn,

}

struct Buyer

{

public string Name;

public short Age;

public bool HasDiscount;

public Buyer(string name, short age, bool discount)

{

Name = name;

Age = age;

HasDiscount = discount;

}

public void ShowInfo()

{

Console.WriteLine($"Покупатель {Name} {Age} \t Скидка: {HasDiscount}");

}

}

}

**Лабораторная 7. Исключения.**

class Program

{

static void Main(string[] args)

{

//lab7 exceptions

Flower testFlower = new Flower("Тюльпан", "Лес", "Желтый", 6, 9.9);

//e1

try

{

testFlower.Cost = 0;

}

catch (FlowerExceptions.CostIsZeroException e)

{

Console.WriteLine("Цена присвоена неверно " + e);

Console.WriteLine(e.Message);

//Console.WriteLine(e.Source);

//Console.WriteLine(e.StackTrace);

//Console.WriteLine(e.TargetSite);

}

//e2

try

{

testFlower.Cost = -15.6;

}

catch (FlowerExceptions.CostLessThanZeroException e)

{

Console.WriteLine(e.Message);

}

//e3

try

{

testFlower.Color = "Черный";

}

catch (FlowerExceptions.CantBeBlackException e)

{

Console.WriteLine(e.Message);

}

//e4

try

{

int a = 0;

int b = 7 / a;

}

catch (DivideByZeroException e)

{

Console.WriteLine(e.Message);

}

//e5

try

{

int[] nums = new int[2];

nums[3] = 15;

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

//e6

try

{

checked

{

int c = int.MaxValue;

c++;

}

}

catch (OverflowException e)

{

Console.WriteLine(e.Message);

}

//try-catch-finally

try

{

int a = 0;

int d = 10 / a;

}

catch (Exception e)

{

Console.WriteLine("Возникла непредвиденная ошибка, описание ошибки:");

Console.WriteLine(e.Message);

Console.WriteLine(e.TargetSite);

Console.WriteLine(e.Source);

Console.WriteLine(e.StackTrace);

}

finally

{

Console.WriteLine("Finally выполняется всегда!");

}

}

}

}

public class FlowerExceptions

{

protected internal class CostLessThanZeroException : Exception

{

public CostLessThanZeroException(string message) : base(message) { }

}

protected internal class CostIsZeroException : Exception

{

public CostIsZeroException(string message) : base(message) { }

}

protected internal class CantBeBlackException : Exception

{

public CantBeBlackException(string message) : base(message) { }

}

}

**Лабораторная 8. Обобщения.**

using System;

using System.Collections.Generic;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

//Сериализация представляет процесс преобразования какого-либо объекта в поток байтов.

namespace Lab8

{

[Serializable]

public partial class GetSet<T> : ICollectionType<T> where T : new()

{

private readonly List<T> storage;

public List<T> Storage

{

get

{

return this.storage;

}

}

public GetSet()

{

this.storage = new List<T>();

}

public GetSet(List<T> storage)

{

this.storage = storage;

}

public void Add(T elem)

{

this.storage.Add(elem);

}

public void Remove(T delem)

{

if (Storage.Contains(delem))

{

this.storage.Remove(delem);

Console.WriteLine("Элемент удалён!");

}

else

{

Console.WriteLine("Элемент не найден!");

}

}

public void Show()

{

foreach (T str in storage)

{

Console.Write(str + " ");

}

Console.WriteLine();

}

public void Search(T selem)

{

if (Storage.Contains(selem))

{

Console.WriteLine("Элемент найден!");

}

else

{

Console.WriteLine("Элемент не найден!");

}

Console.WriteLine();

}

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

public class Plant

{

private string Name;

public string LifeForm { get; set; }

public string this[string str]

{

set

{

Name = value;

}

get

{

return Name;

}

}

public override string ToString()

{

return $"{this.GetType()} {Name} {LifeForm}";

}

}

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

// файл сохраняется в bin\Debug\netcoreapp\*

public void Save(string CurrentFile)

{

BinaryFormatter bf = new BinaryFormatter();

using (FileStream fs = new FileStream(CurrentFile, FileMode.OpenOrCreate))

{

bf.Serialize(fs, Storage);

fs.Close();

}

}

// файл выгружается из bin\Debug\netcoreapp\*

public void Upload(string CurrentFile)

{

BinaryFormatter bf = new BinaryFormatter();

using (FileStream fs = new FileStream(CurrentFile, FileMode.OpenOrCreate))

{

List<T> deser = (List<T>)bf.Deserialize(fs);

foreach (T p in deser)

{

if (p == null)

continue;

this.Add(p);

}

fs.Close();

}

}

}

}

public partial class GetSet<T>

{

public class Owner

{

public Guid ID;

public string Name;

public string Organization;

public Owner()

{

ID = Guid.NewGuid();

Name = "Vasilisa";

Organization = "BSTU";

}

}

public Owner myOwner = new Owner();

public static class Date

{

public static DateTime date = new DateTime(2021, 10, 24);

}

}

interface ICollectionType<T> where T : new()

{

void Add(T elem);

void Remove(T delem);

void Show();

}

using System;

//Обобщение означает параметризированный тип. Особая роль

//параметризированных типов состоит в том, что они позволяют создавать

//классы, структуры, интерфейсы, методы и делегаты, в которых

//обрабатываемые данные указываются в виде параметра

namespace Lab8

{

class Program

{

static void Main(string[] args)

{

try

{

GetSet<object> set1 = new GetSet<object>();

set1.Add(3);

set1.Add(6);

set1.Add(8);

Console.WriteLine($"Множество: ");

set1.Show();

Console.WriteLine("\nИщем элемент 2");

set1.Search(2);

Console.WriteLine("Ищем элемент 3");

set1.Search(3);

Console.WriteLine("Пытаемся удалить элемент 13");

set1.Remove(13);

Console.WriteLine("\nУдаляем элемент 8");

set1.Remove(8);

Console.WriteLine("\nВыводим множество заново:");

set1.Show();

Console.ReadKey();

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

GetSet<Object> set2 = new GetSet<Object>();

set2.Add("Never");

set2.Add("Give");

set2.Add("Up");

set2.Add(".");

set2.Add("You");

set2.Add("Are");

set2.Add("Amasing");

Console.WriteLine($"Множество со строками: ");

set2.Show();

Console.WriteLine("\nИщем слово 'Привет'");

set2.Search("Привет");

Console.WriteLine("Ищем слово 'Up'");

set2.Search("Up");

Console.WriteLine("Пытаемся удалить слово 'Пока'");

set2.Remove("Пока");

Console.WriteLine("\nУдаляем слово 'Are'");

set2.Remove("Are");

Console.WriteLine("\nВыводим множество со строками заново:");

set2.Show();

Console.ReadKey();

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

Console.WriteLine("Сохраняю в файл...");

System.Threading.Thread.Sleep(2000);

set2.Save("saving.txt");

Console.WriteLine("Множество сохранено в файл!");

System.Threading.Thread.Sleep(2000);

Console.WriteLine("Создаю новое множество из записанного файла...");

GetSet<Object> set3 = new GetSet<Object>();

System.Threading.Thread.Sleep(2000);

Console.WriteLine("Готово! Можно посмотреть:");

System.Threading.Thread.Sleep(2000);

set3.Upload("saving.txt");

set3.Show();

}

finally

{

System.Threading.Thread.Sleep(1000);

Console.WriteLine("\nПрограмма завершена!\n");

}

}

}

}

**Лабораторная 9. Делегаты. События. Лямбда-выражения.**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab9

{

public class Director

{

public delegate void Info(string message);

public event Info Notify;

public Director(int pay, string position)

{

Pay = pay;

Position = position;

}

public int Pay { get; private set; }

public string Position { get; private set; }

public override string ToString()

{

return $"Текущая зарплата {Position} составляет {Pay} рублей";

}

// Invoke принимает делегат и выполняет его в том потоке,

// в котором был создан элемент управления, у которого вызывается Invoke.

public void ToPromote(int good)

{

Pay += good;

Notify?.Invoke($"Повышение! Специалисту добавили к зарплате {good} рублей"); // Вызов события + проверка, на null

}

public void Fine(int bad)

{

if (Pay >= bad)

{

Pay -= bad;

Notify?.Invoke($"Штраф: {bad}"); // Вызов события

}

else

{

Notify?.Invoke($"Уволен!"); ;

Pay = 0;

}

}

}

}

using System;

// Делегаты представляют такие объекты, которые указывают на методы.

// То есть делегаты - это указатели на методы

// и с помощью делегатов мы можем вызвать данные методы.

// События сигнализируют системе о том, что произошло определенное действие.

// И если нам надо отследить эти действия, то как раз мы можем применять события.

namespace Lab9

{

class Program

{

public delegate int LambdaExpression(int x, int y); // делегат без параметров

static void Main(string[] args)

{

Director programmer1 = new Director(1400, "Программист\_1");

Director programmer2 = new Director(1000, "Программист\_2");

Director teacher1 = new Director(800, "Учитель\_1");

Director teacher2 = new Director(300, "Учитель\_2");

Director assistant1 = new Director(700, "Ассистент\_1");

Director assistant2 = new Director(200, "Ассистент\_2");

Console.WriteLine(programmer1.ToString());

programmer1.Notify += DisplayGreenMessage;// добавляем обработчик события

programmer1.ToPromote(600);

programmer1.Notify -= DisplayGreenMessage;

Console.WriteLine(programmer1.ToString());

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

Console.WriteLine(teacher2.ToString());

teacher2.Notify += DisplayRedMessage;

teacher2.Fine(400);

teacher2.Notify -= DisplayRedMessage;

Console.WriteLine(teacher2.ToString());

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

Console.WriteLine(assistant1.ToString());

assistant1.Notify += DisplayRedMessage;

assistant1.Fine(400);

assistant1.Notify -= DisplayRedMessage;

assistant1.Notify += DisplayGreenMessage;// добавляем обработчик события

assistant1.ToPromote(600);

assistant1.Notify -= DisplayGreenMessage;

Console.WriteLine(assistant1.ToString());

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

Console.WriteLine("Сумма зарплат Программиста\_1 и Ассистента\_1 при помощи лямбда-выражения:");

LambdaExpression sum = (x, y) => x + y;

Console.WriteLine(sum(programmer1.Pay, assistant1.Pay));

// обработчики событий

static void DisplayGreenMessage(string message)

{

// Устанавливаем красный цвет символов

Console.ForegroundColor = ConsoleColor.Green;

Console.WriteLine(message);

// Сбрасываем настройки цвета

Console.ResetColor();

}

static void DisplayRedMessage(String message)

{

// Устанавливаем красный цвет символов

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(message);

// Сбрасываем настройки цвета

Console.ResetColor();

}

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

Console.ReadKey();

string myString = "Never give up and everything will be OK";

Console.WriteLine($"Делаем все буквы строчными\n{ChangingStrings.TaskToLower(myString)}\n");

Console.WriteLine($"Добавляем в конце точку\n{ChangingStrings.AddChar(myString, '.')}\n");

Console.WriteLine($"Меняем i на I\n{ChangingStrings.Change(myString)}\n");

Console.WriteLine($"Делаем все буквы заглавными\n{ChangingStrings.TaskToUpper(myString)}\n");

Console.WriteLine($"Добавляем в конце восклицательный знак\n{ChangingStrings.AddExclamation(myString)}");

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

Console.ReadKey();

// встроенные делегаты

// Делегат Action является обобщенным, принимает параметры (от 0 до 16) и возвращает значение void

// Делегат Predicate<T> используется для сравнения, сопоставления некоторого объекта T определенному условию.

// Делегат Func возвращает результат действия и может принимать параметры. Также часто используется в качестве параметра в методах.

Console.WriteLine("Обработка строки с использованием стандартных типов делегатов:");

Action<string> Operation = Lower;

Action<string, char> Operation1 = DelegateAddChar;

Func<string, string> Operation2 = Upper;

Operation += Add;

Operation += Chng;

Operation(myString);

Operation1(myString,'.');

Console.WriteLine(Operation2(myString));

Console.WriteLine();

}

// для встроенных делегатов

static void Lower(string a)

{

Console.WriteLine(a.ToLower());

}

static void DelegateAddChar(string a, char letter)

{

Console.WriteLine(a += letter);

}

static void Add(string a)

{

Console.WriteLine(a += "!");

}

static string Upper(string str)

{

return str.ToUpper();

}

static void Chng(string a)

{

Console.WriteLine(a.Replace("i", "I"));

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab9

{

static public class ChangingStrings

{

public static string TaskToLower(string str)

{

str = str.ToLower();

return str;

}

public static string AddChar(string str, char letter)

{

str += letter;

return str;

}

public static string AddExclamation(string str)

{

str += "!";

return str;

}

public static string TaskToUpper(string str)

{

str = str.ToUpper();

return str;

}

public static string Change(string str)

{

str = str.Replace("i", "I");

return str;

}

}

}

**Лабораторная 10. Коллекции.**

using System;

using System.Collections;

namespace Lab10

{

interface IEnumerable<T>

{

void AddElem(T el);

void DelElem(int key);

void Find(int key);

void Print();

}

public class Person

{

public string firstname;

public string lastname;

public Person(string firstname, string lastname)

{

this.firstname = firstname;

this.lastname = lastname;

}

}

// создаем перечисляемую коллекцию работников реализуемую интерфейсом IEnumerable

public class Workers : IEnumerable

{

// создаем массив с элементами типа Person, где коллекция будет хранить свои элементы

private Person[] \_workers;

// конструктор принимает массив в качестве аргумента

public Workers(Person[] pArray)

{

// копируем массив из аргумента в собственный массив коллекции

\_workers = new Person[pArray.Length];

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

{

\_workers[i] = pArray[i];

}

}

// явная реализация интерфейса IEnumerable

IEnumerator IEnumerable.GetEnumerator()

{

// возвращаем перечислитель по коллекции

return (IEnumerator)GetEnumerator();

}

// перечислитель - объект класса WorkersEnum

public WorkersEnum GetEnumerator()

{

// возвращаем перечислитель

return new WorkersEnum(\_workers);

}

// класс перечислителя для коллекции Workers

public class WorkersEnum : IEnumerator

{

// массив с элементами типа Person

public Person[] \_workers;

// (Enumenators позиционированны перед первым элементом)

// определяет позицию перечислителя

int position = -1;

// инициализация массива

public WorkersEnum(Person[] list)

{

\_workers = list;

}

// реализация интерфейсов из IEnumerator

object IEnumerator.Current

{

get

{

return Current;

}

}

public Person Current

{

get

{

// попытка получить элемент коллекции, на который указывает указатель перечислителя

try

{

return \_workers[position];

}

catch (IndexOutOfRangeException)

{

throw new InvalidOperationException();

}

}

}

// перемещает указатель на следующую позицию

// возвращает булевое значение, дошли мы до конца коллекции или нет

public bool MoveNext()

{

position++;

return (position < \_workers.Length);

}

// указатель возвращается в первоначальную позицию

public void Reset()

{

position = -1;

}

}

}

}

using System;

using System.Collections;

namespace Lab10

{

public class Worker: IEnumerable<Worker>

{

private Guid id;

public Guid Id { get => id; }

public string Name { get; set; }

public string Post { get; set; }

public int Pay { get; set; }

public static int Ip = 0;

public int ip = 0;

public Worker(Guid id, string name, string post, int pay)

{

this.id = id;

Name = name;

Post = post;

Pay = pay;

Ip++;

ip = Ip;

}

public Worker()

{

Name = "Неизвестно";

Post = "неизвестно";

Pay = 0;

Ip++;

ip = Ip;

}

Hashtable hashList = new Hashtable();

public void GetHash()

{

Console.WriteLine(GetHashCode());

}

public override string ToString()

{

return $"{Name} - {Post}" +

$" с зарплатой {Pay}" +

$"\nId работника: {Id}\n\n";

}

public void AddElem(Worker el)

{

Ip++;

hashList.Add(Ip, el);

}

public void DelElem(int key)

{

hashList.Remove(key);

}

public void Find(int key)

{

ICollection keys = hashList.Keys;

foreach (int s in keys)

{

if (s == key)

{

Console.WriteLine($"Был найден ключ: {s} со значением {hashList[s]}");

}

else

{

Console.WriteLine("Не найден ключ!");

}

}

}

public void Print()

{

ICollection keys = hashList.Keys;

foreach (int s in keys)

Console.WriteLine(s + ": " + hashList[s]);

}

}

}

using System;

using System.Collections;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

// IEnumerable - интерфейс, предоставляет перечислитель,

// который поддерживает простой перебор элементов неуниверсальной коллекции.

// Коллекция представляет собой совокупность объектов. В среде

// .NET Framework имеется немало интерфейсов и классов, в которых

// определяются и реализуются различные типы коллекций.

// Главное преимущество коллекций заключается в том, что они

// стандартизируют обработку групп объектов в программе.

namespace Lab10

{

class Program

{

static void Main(string[] args)

{

#region [ The first task ]

Console.WriteLine("\t\t\t\tЗадание 1:\n");

Person[] personArray =

{

new Person("Владимир","Распутин"),

new Person("Михаил","Ждарин"),

new Person("Виталий","Квазинский")

};

Workers workersCollection = new Workers(personArray);

foreach (var person in workersCollection)

{

Console.WriteLine("Имя работника: {0}, фамилия работника: {1}", person.firstname, person.lastname);

}

Console.WriteLine("Количество элементов: " + personArray.Length);

Console.ReadKey();

#endregion

#region [ The second task ]

Console.WriteLine("\t\t\t\tЗадание 2:\n");

Worker obj1 = new Worker(Guid.NewGuid(), "Павел Викторович", "декан", 3000);

Worker obj2 = new Worker(Guid.NewGuid(), "Николай Львович", "ректор", 16030);

Worker obj3 = new Worker(Guid.NewGuid(), "Иван Кузьмич", "лектор", 2400);

Worker obj4 = new Worker(Guid.NewGuid(), "Валерий Владимирович", "лаборант", 1400);

Worker obj5 = new Worker(Guid.NewGuid(), "Арина Николаевна", "лаборант", 1500);

Worker obj6 = new Worker(Guid.NewGuid(), "Надежда Александровна", "лектор", 1900);

Console.WriteLine($"{obj1.ToString()}\n" +

$"{obj2.ToString()}\n" +

$"{obj3.ToString()}\n" +

$"{obj4.ToString()}\n" +

$"{obj5.ToString()}\n" +

$"{obj6.ToString()}\n");

Hashtable workers = new Hashtable();

workers.Add(obj1.ip, obj1);

workers.Add(obj2.ip, obj2);

workers.Add(obj3.ip, obj3);

workers.Add(obj4.ip, obj4);

workers.Add(obj5.ip, obj5);

workers.Add(obj6.ip, obj6);

ICollection keys = workers.Keys;

foreach (int n in keys)

{

Console.WriteLine(n + ": " + workers[n]);

}

Console.ReadKey();

#endregion

#region [ The third task ]

//Представляет коллекцию динамических данных, обеспечивающих выдачу уведомлений при получении и удалении элементов или при обновлении всего списка. Коллекция предназначена для того, чтобы пользовательский интерфейс мог получать информацию об изменениях коллекции. Используется в случае если нужна информация о том, когда элементы коллекции удаляются или добавляются.

Console.WriteLine("\t\t\t\tЗадание 3\n");

ObservableCollection<Worker> p = new ObservableCollection<Worker>();

void p\_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)

{

switch (e.Action)

{

case NotifyCollectionChangedAction.Add:

Worker newWorker = e.NewItems[0] as Worker;

Console.WriteLine("Добавлен новый объект: " + newWorker.Name);

break;

case NotifyCollectionChangedAction.Remove:

Worker oldWorker = e.OldItems[0] as Worker;

Console.WriteLine("Удален объект: " + oldWorker.Name);

break;

}

}

p.CollectionChanged += p\_CollectionChanged;

p.Add(obj1);

p.Add(obj2);

p.Add(obj3);

p.RemoveAt(1);

Console.WriteLine("\n");

foreach (Worker i in p)

{

Console.WriteLine(i.ToString());

}

#endregion }}}

**Лабораторная 11. LINQ.**

using System;

namespace Lab11

{

public class Player

{

public string Name { get; set; }

public string Team { get; set; }

}

public class Game

{

public string Name { get; set; }

public string Developer { get; set; }

}

public partial class Customer //Частичные классы (partial classes) – конструкция, которая позволяет определить один класс в нескольких файлах.

{

public static string about = "Этот класс о клиентах.";

private static int count; //Ключевое слово static - объявление константы или типа неявно является членом static. На член static невозможно ссылаться через экземпляр, а можно только через имя типа.

private string name;

private int balance;

// Свойства

public string Name { get => name; set { name = value; } }

public int Balance { get => balance; set { balance = value; } }

//Конструкторы

// с параметрами

public Customer(string name, int balance)

{

this.name = name;

this.balance = balance;

Customer.count++;

}

// без параметров

public Customer()

{

name = "Николай";

balance = 88727;

}

// с параметрами по умолчанию

public Customer(int balance, string name = "Игорь")

{

this.name = name;

this.balance = balance;

}

public override string ToString() // переопределение метода ToString

{

return $"Имя: {this.name}\n" +

$"Баланс: {this.balance} рублей\n";

}

// Метод вывода информации о классе

public static string GetInfoAboutClass()

{

return about;

}

public void PrintIt()

{

Console.WriteLine(this.ToString());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace Lab11

{

class Program

{

static Random rnd = new Random();

static void Main(string[] args)

{

#region [The first task]

Console.WriteLine("\t\t\t\tЗадание 1");

string[] year = { "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", "December" };

Console.WriteLine("\t\t\tСортировка по алфавиту");

var alphabeth = from month in year orderby month select month;

foreach (string months in alphabeth)

{

Console.WriteLine(months);

}

Console.WriteLine("\n\t\tВыборка из месяцев по длине строки, равной 4");

int n = 8;

var length = from month in year where month.Length == n select month;

foreach (string months in length)

{

Console.WriteLine(months);

}

Console.WriteLine("\n\t\t\tВыборка из месяцев с буквой 'u' ");

var u = from month in year where month.Contains('u') && month.Length >= 4 select month;

foreach (string months in u)

{

Console.WriteLine(months);

}

Console.WriteLine("\n\t\tВыборка из месяцев: зимние и летние месяцы");

var WinterSummer = from month in year

where month == year[0] || month == year[1] || month == year[7] || month == year[11] || month == year[5] || month == year[6]

select month;

foreach (string months in WinterSummer)

{

Console.WriteLine(months);

}

Console.ReadKey();

#endregion

#region [The second task]

Console.WriteLine("\t\t\t\tЗадание 2");

var collection = new List<Customer>();

for (char i = 'А'; i != 'Ж'; i++)

{

var customer = new Customer

{

Name = "Покупатель " + i,

Balance = rnd.Next(10, 2500)

};

collection.Add(customer);

}

// вывод всей коллекции

foreach (var item in collection)

{

Console.WriteLine(item);

}

Console.WriteLine("\tСписок покупателей, расположенных по возрастанию баланса: ");

var sortCollectionByCash = collection.OrderBy(item => item.Balance);

foreach (var item in sortCollectionByCash)

{

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

}

Console.WriteLine("\tСписок покупателей, у которых баланс больше, чем 500 рублей," +

"\n\t\t\tно меньше, чем 1500: \n");

var sortCollectionByBalance = collection.Where(item => item.Balance < 1500 && item.Balance > 500).OrderBy(item => item.Balance);

foreach (var item in sortCollectionByBalance)

{

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

}

Console.WriteLine("\t\t\tМаксимальный покупатель (по балансу): ");

int max = collection.Max(a => a.Balance);

var sortCollectionMax = collection.Where(item => item.Balance == max).OrderBy(item => item.Balance);

foreach (var item in sortCollectionMax)

{

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

}

Console.WriteLine("\tПервые пять покупателей с максимальной суммой на карте: ");

var sortCollection5 = collection.Where(item => item.Balance <= max).OrderBy(item => item.Balance);

var unsortedCollection5 = sortCollection5.Reverse();

var sortedCollection5 = unsortedCollection5.Take(5);

foreach (var item in sortedCollection5)

{

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

}

Console.ReadKey();

#endregion

#region [The third task]

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

{ new Game { Name = "DOTA2", Developer ="Valve" },

new Game { Name = "Genshin Impact", Developer ="miHoYo" }

};

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

{

new Player {Name="Василиса", Team="DOTA2"},

new Player {Name="Полина", Team="DOTA2"},

new Player {Name="Лиза", Team="Genshin Impact"}

};

var result = from pl in players

join t in teams on pl.Team equals t.Name

select new { Name = pl.Name, Team = pl.Team, Country = t.Developer };

foreach (var item in result)

{

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

}

#endregion

}

}}

**Лабораторная 12. Рефлексия.**

using System;

using System.IO;

using System.Reflection; // Customer в этом листинге из третьей лабы

namespace Lab12

{

static class Reflector

{

// Метод записи в файл

static void Write(string text)

{

string path = "output.txt";

File.AppendAllText(path, text);

}

// Определение имени сборки, в которой определен класс

public static void GetAssemblyName<T>(T obj)

{

Assembly a = typeof(T).Assembly; // класс, представляющий сборку и позволяющий манипулировать этой сборкой

Write("Имя сборки, в которой определен класс: " + a.FullName + "\n\n");

Console.WriteLine("Имя сборки, в которой определен класс: " + a.FullName + "\n");

}

// MemberInfo: базовый абстрактный класс,

// определяющий общий функционал для классов EventInfo, FieldInfo, MethodInfo и PropertyInfo

// Есть ли публичные конструкторы?

public static void IsPublicConstructors<T>(T obj)

{

ConstructorInfo[] constructorInfo = typeof(T).GetConstructors(); //класс, представляющий конструктор

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

if ((constructorInfo[i].IsPublic))

{

Console.WriteLine("Класс Worker имеет публичный конструктор " + (i+1) + "\n");

Write("Класс Worker имеет публичный конструктор " + (i+1) + "\n");

}

else

{

Console.WriteLine("Класс Worker не имеет публичный конструктор \n");

Write("Класс Worker не имеет публичный конструктор\n");

}

}

// Извлекает все общедоступные публичные методы класса

public static void GetAllMethods<T>(T obj)

{

MethodInfo[] methodInfo = typeof(T).GetMethods(); // хранит информацию об определенном методе

Console.Write($"Извлекает все общедоступные публичные методы класса: \n");

Write($"\nИзвлекает все общедоступные публичные методы класса: \n");

foreach (MethodInfo method in methodInfo)

{

string modificator = "";

if (method.IsStatic)

modificator += "static ";

if (method.IsVirtual)

modificator += "virtual";

Console.Write($"{modificator} {method.ReturnType.Name} {method.Name} \n");

Write($"{modificator} {method.ReturnType.Name} {method.Name} \n");

}

}

// Информация о полях и свойствах класса

public static void GetFieldsProp<T>(T obj)

{

FieldInfo[] fieldInfo = typeof(T).GetFields(); // хранит информацию об определенном поле типа

PropertyInfo[] propertyInfo = typeof(T).GetProperties(); // хранит информацию о свойстве

Console.WriteLine("\nИнформация о полях:");

Write("\nИнформация о полях:\n");

foreach (FieldInfo field in fieldInfo)

{

Console.WriteLine($"{field.FieldType} {field.Name}");

Write($"{field.FieldType} {field.Name}\n");

}

Console.WriteLine("\nСвойства класса:");

Write("\nСвойства класса:\n");

foreach (PropertyInfo prop in propertyInfo)

{

Console.WriteLine($"{prop.PropertyType} {prop.Name}\n");

Write($"{prop.PropertyType} {prop.Name}\n");

}

}

// Получаем все реализованные классом интерфейсы

public static void GetAllInterfaces<T>(T obj)

{

Type[] interfacesInfo = typeof(T).GetInterfaces();

Console.WriteLine("Все реализованные классом интерфейсы:");

Write("\nВсе реализованные классом интерфейсы:\n");

foreach (Type i in interfacesInfo)

{

Console.WriteLine(i.Name);

Write(i.Name);

}

}

// Выводит по имени класса имена методов, которые содержат заданный пользователем тип параметра

public static void GetSpecialMethod<T>(T obj, string type)

{

MethodInfo[] methodInfo = typeof(T).GetMethods();

Console.WriteLine("Выводит по имени класса имена методов, которые содержат заданный пользователем тип параметра:");

Write("\n\nВыводит по имени класса имена методов, которые содержат заданный пользователем тип параметра:\n");

foreach (MethodInfo method in methodInfo)

{

ParameterInfo[] parameters = method.GetParameters();

string modificator = "";

if (method.IsStatic)

modificator += "static ";

if (method.IsVirtual)

modificator += "virtual";

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

{

if (parameters[i].ParameterType.Name == type)

{

Console.Write($"{modificator} {method.ReturnType.Name} {method.Name} \n");

Write($"{modificator} {method.ReturnType.Name} {method.Name}\n");

break;

}

}

}

}

// Метод Invoke, который вызывает метод класса

// 1) прочитать из текстового файла (имя класса и имя метода передаются в качестве аргументов)

// 2) сгенерировать, используя генератор значений для каждого типа.

public static void Invoke<S>(string methodName, string className, S parametrs)

{

//Чтобы динамически загрузить сборку в приложение, надо использовать статические методы Assembly.LoadFrom() или Assembly.Load().

//Метод LoadFrom() принимает в качестве параметра путь к сборке.

//Метод Load() в качестве его параметра передается дружественное имя сборки, которое нередко совпадает с именем приложения.

Assembly asm = Assembly.LoadFrom("Lab12.dll");

Type t = asm.GetType("Lab12." + className, true, true);

object obj = Activator.CreateInstance(t);

MethodInfo method = t.GetMethod(methodName);

object result = method.Invoke(obj, new object[] { parametrs });

Console.WriteLine(result);

}

// Метод Create создает объект переданного типа

public static object Create(string className)

{

Assembly asm = Assembly.LoadFrom("Lab12.dll");

Type t = asm.GetType("Lab12." + className, true, true);

return Activator.CreateInstance(t);

}

}

}

using System;

// Рефлексия - процесс выявления типов во время выполнения приложения.

namespace Lab12

{

class Program

{

static void Main()

{

Worker obj1 = new Worker(Guid.NewGuid(), "Павел Викторович", "декан", 3000);

Customer obj2 = new Customer(Guid.NewGuid(), "Игоренко", "Николай", "Васильевич", "г.Могилёв, ул. Крестьянская, д. 21, кв. 52", 7340930482768764, 36229);

Console.WriteLine("Работает рефлектор...\n");

Reflector.GetAssemblyName(obj1);

Reflector.IsPublicConstructors(obj1);

Reflector.GetAllMethods(obj1);

Reflector.GetFieldsProp(obj1);

Reflector.GetAllInterfaces(obj1);

Reflector.GetSpecialMethod(obj2, "String");

Reflector.Invoke("Repeat", "Customer", "\nТЕСТИРОВКА");

Console.WriteLine(Reflector.Create("Customer").ToString());

}

}

}

using System;

using System.Collections;

namespace Lab12

{

interface IEnumerable<T>

{

void AddElem(T el);

void DelElem(int key);

void Find(int key);

void Print();

}

public class Person

{

public string firstname;

public string lastname;

public Person(string firstname, string lastname)

{

this.firstname = firstname;

this.lastname = lastname;

}

}

// создаем перечисляемую коллекцию работников реализуемую интерфейсом IEnumerable

public class Workers : IEnumerable

{

// создаем массив с элементами типа Person, где коллекция будет хранить свои элементы

private Person[] \_workers;

// конструктор принимает массив в качестве аргумента

public Workers(Person[] pArray)

{

// копируем массив из аргумента в собственный массив коллекции

\_workers = new Person[pArray.Length];

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

{

\_workers[i] = pArray[i];

}

}

// явная реализация интерфейса IEnumerable

IEnumerator IEnumerable.GetEnumerator()

{

// возвращаем перечислитель по коллекции

return (IEnumerator)GetEnumerator();

}

// перечислитель - объект класса WorkersEnum

public WorkersEnum GetEnumerator()

{

// возвращаем перечислитель

return new WorkersEnum(\_workers);

}

// класс перечислителя для коллекции Workers

public class WorkersEnum : IEnumerator

{

// массив с элементами типа Person

public Person[] \_workers;

// (Enumenators позиционированны перед первым элементом)

// определяет позицию перечислителя

int position = -1;

// инициализация массива

public WorkersEnum(Person[] list)

{

\_workers = list;

}

// реализация интерфейсов из IEnumerator

object IEnumerator.Current

{

get

{

return Current;

}

}

public Person Current

{

get

{

// попытка получить элемент коллекции, на который указывает указатель перечислителя

try

{

return \_workers[position];

}

catch (IndexOutOfRangeException)

{

throw new InvalidOperationException();

}

}

}

// перемещает указатель на следующую позицию

// возвращает булевое значение, дошли мы до конца коллекции или нет

public bool MoveNext()

{

position++;

return (position < \_workers.Length);

}

// указатель возвращается в первоначальную позицию

public void Reset()

{

position = -1;

}

}

}

}

using System;

using System.Collections;

namespace Lab12

{

public class Worker : IEnumerable<Worker>

{

private Guid id;

public Guid Id { get => id; }

public string Name { get; set; }

public string Post { get; set; }

public int Pay { get; set; }

public static int Ip = 0;

public int ip = 0;

public Worker(Guid id, string name, string post, int pay)

{

this.id = id;

Name = name;

Post = post;

Pay = pay;

Ip++;

ip = Ip;

}

public Worker()

{

Name = "Неизвестно";

Post = "неизвестно";

Pay = 0;

Ip++;

ip = Ip;

}

Hashtable hashList = new Hashtable();

public void GetHash()

{

Console.WriteLine(GetHashCode());

}

public override string ToString()

{

return $"{Name} - {Post}" +

$" с зарплатой {Pay}" +

$"\nId работника: {Id}\n\n";

}

public void AddElem(Worker el)

{

Ip++;

hashList.Add(Ip, el);

}

public void DelElem(int key)

{

hashList.Remove(key);

}

public void Find(int key)

{

ICollection keys = hashList.Keys;

foreach (int s in keys)

{

if (s == key)

{

Console.WriteLine($"Был найден ключ: {s} со значением {hashList[s]}");

}

else

{

Console.WriteLine("Не найден ключ!");

}

}

}

public void Print()

{

ICollection keys = hashList.Keys;

foreach (int s in keys)

Console.WriteLine(s + ": " + hashList[s]);

}

}

}

**Лабораторная 13. Потоки. Файловая система.**

using System;

using System.IO;

using System.IO.Compression;

using System.Text;

namespace Lab13

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\t\t\t\tЗадание 1");

Console.WriteLine("Смотреть выполнение задания в файле VKLog.txt");

VKLog log = new VKLog();

VKForFileInfo fileInfo = new VKForFileInfo();

log.WriteFile(fileInfo.FullPath());

log.WriteFile(fileInfo.Info());

log.WriteFile(fileInfo.Dates());

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

VKDiskInfo.DiskInfo();

VKFileInfo.FileInfo("VKlog.txt");

VKDirInfo.DirInfo(@"D:\Учеба в БГТУ\Пацей Н.В. ООП");

VKFileManager.AllAboutDrive("D");

VKFileManager.FileDirCreate("VKInspect", "VKdirinfo", "testFileCopy");

VKFileManager.VKFiles(@"D:\Учеба в БГТУ\Пацей Н.В. ООП\Лабораторная 13", "xlsx");

VKFileManager.Zip();

VKFileManager.UnZip(@"D:\VKInspect\UnZip");

}

}

}

static class VKDirInfo

{

public static void DirInfo(string dirName)

{

Console.WriteLine("\t\t\t\tЗадание 4");

DirectoryInfo dir = new DirectoryInfo(dirName);

Console.WriteLine($"Название каталога: {dir.Name}");

Console.WriteLine($"Полный путь каталога: {dir.FullName}");

Console.WriteLine($"Время создания каталога: {dir.CreationTime}");

Console.WriteLine($"Количестве файлов: {dir.GetFiles().Length}");

Console.WriteLine($"Количестве поддиректориев: {dir.GetDirectories().Length}");

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

}

}

static class VKDiskInfo

{

public static void DiskInfo()

{

Console.WriteLine("\t\t\t\tЗадание 2");

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

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

Console.WriteLine($"Файловая система: {drive.DriveFormat}");

Console.WriteLine($"Свободное место на диске: {Math.Round(drive.AvailableFreeSpace \* 1e-9, 2)} ГБайт");

Console.WriteLine($"Всего места на диске: {Math.Round(drive.TotalSize \* 1e-9, 2)} ГБайт");

Console.WriteLine($"Метка: {drive.VolumeLabel}\n");

}

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

}

}

static class VKFileInfo

{

public static void FileInfo(string path)

{

Console.WriteLine("\t\t\t\tЗадание 3");

FileInfo file = new FileInfo(path);

Console.WriteLine($"Полный путь: {file.FullName}");

Console.WriteLine($"Имя файла: {file.Name}");

Console.WriteLine($"Расширение: {file.Extension}");

Console.WriteLine($"Размер: {file.Length}");

Console.WriteLine($"Дата создания: {file.CreationTime}\n");

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

}

}

class VKForFileInfo

{

private string action;

public string FullPath(string path = @"D:\Учеба в БГТУ\Пацей Н.В. ООП\Лабораторная 13\Lab13\Lab13\bin\Debug\netcoreapp3.1\VKLog.txt")

{

FileInfo f = new FileInfo(path);

action = "Полный путь к файлу: " + f.DirectoryName;

return action;

}

public string Info(string path = @"D:\Учеба в БГТУ\Пацей Н.В. ООП\Лабораторная 13\Lab13\Lab13\bin\Debug\netcoreapp3.1\VKLog.txt")

{

FileInfo f = new FileInfo(path);

action = "Размер: " + f.Length + "байт. ";

action += "Расширение: " + f.Extension + ". ";

action += "Имя: " + f.FullName + ". ";

return action;

}

public string Dates(string path = @"D:\Учеба в БГТУ\Пацей Н.В. ООП\Лабораторная 13\Lab13\Lab13\bin\Debug\netcoreapp3.1\VKLog.txt")

{

FileInfo f = new FileInfo(path);

action = "Дата создания: " + f.CreationTime + ". ";

action += "Дата изменения: " + f.LastWriteTime + ". ";

return action;

}

}

static class VKFileManager

{

public static void AllAboutDrive(string disk)

{

Console.WriteLine("\t\t\t\tЗадание 5");

string path;

if (disk.ToLower() == "d")

{

path = @"D:\";

}

else

{

throw new Exception("Неверный ввод.");

}

DirectoryInfo dir = new DirectoryInfo(path);

Console.WriteLine("\nСписок папок диска:");

foreach (DirectoryInfo name in dir.GetDirectories())

{

Console.WriteLine(name.Name);

}

Console.WriteLine("\nСписок файлов диска:");

foreach (FileInfo name in dir.GetFiles())

{

Console.WriteLine(name.Name);

}

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

}

public static void FileDirCreate(string dirName, string fileName, string fileName2)

{

string dirPath = @"D:\" + dirName;

DirectoryInfo dirInfo = new DirectoryInfo(dirPath);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

string filePath = dirPath + "\\" + fileName + ".txt";

FileInfo fileInf = new FileInfo(filePath);

if (!fileInf.Exists)

{

fileInf.Create();

}

try

{

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

sw.WriteLine("Информация!!!");

sw.Close();

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

string filePath2 = dirPath + "\\" + fileName2 + ".txt";

File.Copy(filePath, filePath2, true);

File.Delete(filePath);

}

public static void VKFiles(string path, string ext)

{

string dirPath = @"D:\VKFiles";

DirectoryInfo dirInfo = new DirectoryInfo(dirPath);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

DirectoryInfo userDirInfo = new DirectoryInfo(path);

foreach (FileInfo file in userDirInfo.GetFiles())

{

if (file.Extension == ("." + ext))

file.CopyTo(dirPath + "\\" + file.Name, true);

}

dirInfo.MoveTo(@"D:\VKInspect\VKFiles");

}

public static void Zip()

{

DirectoryInfo source = new DirectoryInfo(@"D:\VKInspect\VKFiles");

ZipFile.CreateFromDirectory(source.FullName, source.FullName + ".zip");

Console.WriteLine("Архив создан!");

}

public static void UnZip(string FolderName)

{

DirectoryInfo source = new DirectoryInfo(@"D:\VKInspect\VKFiles");

ZipFile.ExtractToDirectory(source.FullName + ".zip", FolderName);

Console.WriteLine("Архив разархивирован!");

}

}

public class VKLog

{

readonly DateTime DateNow = DateTime.Now;

public void WriteFile(string action, string path = @"D:\Учеба в БГТУ\Пацей Н.В. ООП\Лабораторная 13\Lab13\Lab13\bin\Debug\netcoreapp3.1\VKLog.txt", string fileName = "VKLog.txt")

{

using (StreamWriter sw = new StreamWriter(path, true, Encoding.UTF8))

{

sw.WriteLine("Имя файла: " + fileName);

sw.WriteLine("Путь к файлу " + path);

sw.WriteLine("Дата создания: " + DateNow);

sw.WriteLine(action);

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

}

}

private string actionOne;

public string FreeSpace()

{

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

actionOne = "Свободное место на диске в байтах: " + drive.AvailableFreeSpace.ToString();

}

return actionOne;

}

public string FileSystem()

{

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

actionOne = "Файловая система: " + drive.DriveFormat.ToString();

}

return actionOne;

}

public string Disk()

{

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

actionOne = "Имя диска: " + drive.Name + ". ";

actionOne += "Объём: " + drive.TotalFreeSpace + "байт. ";

actionOne += "Доступный объём: " + drive.AvailableFreeSpace + "байт. ";

actionOne += "Метка тома: " + drive.VolumeLabel + ". ";

}

return actionOne;

}

}

**Лабораторная 14. Сериализация.**

using System;

using System.Xml.Serialization;

namespace Lab14

{

[Serializable]

[XmlRoot("Human")]

public partial class Customer : ICustomer

{

private string sex;

private int age;

private int iq;

private string name;

private int lifelength;

public int LifeLength

{

get => lifelength;

set => lifelength = value;

}

public Customer()

{

LifeLength = 100;

sex = "";

age = -1;

iq = -1;

name = "";

}

public Customer(string \_sex, int \_age, int \_iq, string \_name)

{

LifeLength = 100;

sex = \_sex;

age = \_age;

iq = \_iq;

name = \_name;

}

public string Name

{

get => name;

set => name = value;

}

public string Sex

{

get => sex;

set => sex = value;

}

public int Age

{

get => age;

set => age = value;

}

public int IQ

{

get => iq;

set => iq = value;

}

public void Think()

{

Console.WriteLine("think");

}

public override string ToString()

{

return "Покупатель:\n" +

"Продолжительность жизни: " + "~" + LifeLength + " лет" +

"\nИмя: " + name.ToString() +

"\nПол: " + sex.ToString() +

"\nВозвраст: " + age.ToString() + " лет" +

"\nIQ: " + iq.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Xml.Serialization;

using System.Xml;

namespace Lab14

{

[Serializable]

[XmlRoot("Human")]

public class GroupOfCustomers

{

private List<Customer> units = new List<Customer>();

public GroupOfCustomers()

{

}

[XmlAttribute]

public int Count => units.Count;

[XmlArray("ListofUnits")]

[XmlArrayItem("Human")]

[XmlIgnore]

public List<Customer> ListofUnits

{

get => units;

set => units = value;

}

public void Add(Customer human)

{

Customer x = human;

units.Add(x);

}

public void Remove(Customer human)

{

Customer x = human;

units.Remove(x);

}

public void ToConsole()

{

Customer[] arr = units.ToArray();

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

{

Console.WriteLine((i + 1).ToString() + ' ' + arr[i].ToString() + '\n');

}

}

}

}

public interface ICustomer

{

int LifeLength { get; }

int IQ { get; }

string Name { get; }

void Think();

}

using System;

using System.Linq;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization.Formatters.Soap;

using System.Xml.Serialization;

using System.Runtime.Serialization.Json;

using System.Xml.Linq;

using System.Xml.XPath;

namespace Lab14

{

class Program

{

static void Main(string[] args)

{

Customer human = new Customer("Мужчина", 21, 150, "Адам");

Customer human1 = new Customer("Женщина", 20, 149, "Ева");

GroupOfCustomers army = new GroupOfCustomers();

army.Add(human);

army.Add(human1);

BinaryFormatter formatter = new BinaryFormatter();

SoapFormatter formatter1 = new SoapFormatter();

XmlSerializer formatter2 = new XmlSerializer(typeof(GroupOfCustomers));

DataContractJsonSerializer formatter3 = new DataContractJsonSerializer(typeof(GroupOfCustomers));

using (FileStream fs = new FileStream("groupOfCust.dat", FileMode.OpenOrCreate))

{

formatter.Serialize(fs, army);

}

using (FileStream fs = new FileStream("customer.soap", FileMode.OpenOrCreate))

{

formatter1.Serialize(fs, human);

}

using (TextWriter fs = new StreamWriter("groupOfCust.xml"))

{

formatter2.Serialize(fs, army);

}

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

{

formatter3.WriteObject(fs, army);

}

using (FileStream fs = new FileStream("groupOfCust.dat", FileMode.OpenOrCreate))

{

GroupOfCustomers army1;

army1 = (GroupOfCustomers)formatter.Deserialize(fs);

Console.WriteLine("Binary\n");

army1.ToConsole();

}

using (FileStream fs = new FileStream("customer.soap", FileMode.OpenOrCreate))

{

Customer human3;

human3 = (Customer)formatter1.Deserialize(fs);

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

Console.WriteLine(human3.ToString());

}

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

{

GroupOfCustomers army1;

army1 = (GroupOfCustomers)formatter2.Deserialize(fs);

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

army1.ToConsole();

}

Customer[] humans = new Customer[] { new Customer("Мужчина", 18, 121, "Авель"), new Customer("Женщина", 15, 123, "Аклима"), new Customer("Мужчина", 15, 180, "Каиф") };

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

{

formatter2 = new XmlSerializer(typeof(Customer[]));

formatter2.Serialize(fs, humans);

}

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

{

foreach (Customer x in (Customer[])formatter2.Deserialize(fs))

Console.WriteLine(x.ToString());

}

XPathDocument xmldoc = new XPathDocument("array.xml");

Console.WriteLine("HERE");

foreach (XPathItem x in xmldoc.CreateNavigator().Select("//Human/Name"))

Console.WriteLine(x.Value);

Console.WriteLine();

foreach (XPathItem x in xmldoc.CreateNavigator().Select("//Human[Sex = \"Man\"]/Name"))

Console.WriteLine(x.Value);

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

{

XDocument xdoc = XDocument.Load(fs);

XElement root = xdoc.Element("ArrayOfCustomer");

foreach (XElement xe in root.Elements("Human").Where(x => x.Element("Name").Value.Contains("а")).ToList())

{

if (Int32.Parse(xe.Element("Age").Value) > 19)

{

xe.Add(new XElement("Permission", "Yes"));

}

else {

xe.Remove(); }

}

root.Add(new XElement("Human",

new XElement("LifeLength", "100"),

new XElement("Name", "Аменадиил"),

new XElement("Sex", "Мужчина"),

new XElement("Age", "20"),

new XElement("IQ", "200"),

new XElement("Permission", "Yes")));

xdoc.Save("newArray.xml");

}

}

}

}

**Лабораторная 15. Работа с потоками выполнения.**

using System;

using System.Reflection;

using System.Diagnostics;

using System.Threading;

using System.IO;

using static System.Console;

// Процесс — экземпляр программы во время выполнения, независимый объект,

//которому выделены системные ресурсы (например, процессорное время и память).

// Поток — определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса,

//это изменение сразу же становится видно другим потокам этого процесса.

// В .NET исполняемые файлы не обслуживаются прямо внутри процесса Windows.

//Они обслуживаются в отдельном логическом разделе внутри процесса,

//который называется доменом приложения (Application Domain — AppDomain).

//В процессе может содержаться несколько доменов приложений.

namespace Lab15

{

#region [ The fourth task ]

public class MyNumbers

{

private int n;

public StreamWriter file = new StreamWriter("result.txt", false);

public MyNumbers(int \_n)

{

n = \_n;

}

~MyNumbers()

{

try { file.Close(); }

catch { };

}

public void OddNumbers() // нечетные

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

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

{

Thread.Sleep(30);

if (i % 2 != 0)

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

file.WriteLine(i);

WriteLine("Нечетное число " + i);

}

}

if (file.BaseStream != null)

file.Close();

}

public void EvenNumbers() // четные

{

lock (this)

// Как только поток войдет в контекст lock, маркер блокировки (в данном случае — текущий объект)

// станет недоступным другим потокам до тех пор, пока блокировка не будет снята по выходе из контекста lock.

{

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

{

if (i % 2 == 0)

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

file.WriteLine(i);

WriteLine("Четное число " + i);

}

if (file.BaseStream != null)

file.Close();

}

}

}

public void OddNumbers1()

{

Monitor.Enter(this);

// Наряду с оператором lock для синхронизации потоков мы можем использовать мониторы

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

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

{

if (i % 2 != 0)

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

file.WriteLine(i);

WriteLine("Нечетное " + i);

Monitor.Pulse(this);

Monitor.Wait(this);

}

}

if (file.BaseStream != null)

file.Close();

Monitor.Exit(this);

}

}

public void EvenNumbers1()

{

Monitor.Enter(this);

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

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

{

if (i % 2 == 0)

{

if (file.BaseStream == null)

file = new StreamWriter("result.txt", true);

file.WriteLine(i);

WriteLine("Четное " + i);

Monitor.Pulse(this);

Monitor.Wait(this);

}

}

if (file.BaseStream != null)

file.Close();

Monitor.Exit(this);

}

}

public void ForTimer(object obj)

{

Console.Clear();

Console.WriteLine("\u2665 Текущее время: " + DateTime.Now.ToLongTimeString() + " \u2665");

}

}

#endregion

class Program

{

#region [ The third task ]

public static void begin(object n)

{

WriteLine("Имя потока: " + Thread.CurrentThread.Name);

WriteLine("Приоритет потока: " + Thread.CurrentThread.Priority.ToString());

WriteLine("Id потока: " + Thread.CurrentThread.ManagedThreadId.ToString());

WriteLine("Статус потока: " + Thread.CurrentThread.ThreadState.ToString());

StreamWriter fileOut = new StreamWriter("myResult.txt");

bool check;

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

{

Thread.Sleep(10);

check = true;

for (int j = i; j > 0; j--)

if ((i % j == 0) && (j != 1) && (j != i))

{

check = false;

break;

}

if (check == true)

{

fileOut.WriteLine(i.ToString());

WriteLine("Простые числа " + i.ToString());

}

}

fileOut.Close();

}

#endregion

static void Main(string[] args)

{

#region [ The first task ]

Process[] proc = Process.GetProcesses();

foreach (Process p in proc)

{

try

{

WriteLine("Id процесса: " + p.Id.ToString());

WriteLine("Имя процесса: " + p.ProcessName);

WriteLine("Приоритет процесса: " + p.BasePriority.ToString() + " " + p.PriorityClass.ToString());

WriteLine("Время запуска: " + p.StartTime.ToString());

WriteLine("Время использования процессором: " + p.TotalProcessorTime.ToString());

if (p.StartTime.ToString() != null)

WriteLine("Текущее состояние: запущен");

WriteLine();

}

catch

{

WriteLine();

}

}

#endregion

#region [ The second task ]

AppDomain domain = AppDomain.CurrentDomain;

WriteLine($"Имя домена: {domain.FriendlyName}");

WriteLine($"Базовая директория: {domain.BaseDirectory}");

WriteLine($"Информация: {domain.SetupInformation}");

WriteLine();

Assembly buf = null;

foreach (Assembly x in domain.GetAssemblies())

{

if (x.GetName().Name == "Lab15")

buf = x;

WriteLine(x.ToString());

}

AppDomain myNewDomain = AppDomain.CreateDomain("Мой новый домен");

Assembly buf2 = myNewDomain.Load(buf.GetName());

AppDomain.Unload(myNewDomain);

WriteLine(buf2.ToString());

#endregion

#region [The third task]

int n;

bool check;

while (true)

{

WriteLine("Введите n:");

check = int.TryParse(ReadLine(), out n);

if (check == false)

WriteLine("Неверно введено значение. Введите снова");

else break;

}

Thread myThread = new Thread(new ParameterizedThreadStart(begin));

myThread.Name = "Абсолютно новый поток";

myThread.Priority = ThreadPriority.Highest;

myThread.Start(n);

#endregion

#region [The fourth task]

Thread thread1, thread2;

MyNumbers numbers = new MyNumbers(n);

thread1 = new Thread(numbers.OddNumbers);

thread2 = new Thread(numbers.EvenNumbers);

thread1.IsBackground = true;

thread2.IsBackground = true;

thread1.Priority = ThreadPriority.AboveNormal;

thread1.Start();

thread2.Start();

Thread.Sleep(900);

Thread thread3 = new Thread(numbers.OddNumbers1);

Thread thread4 = new Thread(numbers.EvenNumbers1);

thread3.IsBackground = true;

thread4.IsBackground = true;

thread3.Priority = ThreadPriority.AboveNormal;

thread3.Start();

thread4.Start();

#endregion

#region [The fifth task]

Thread.Sleep(600);

WriteLine("\n\t\tТАЙМЕР! СЕЙЧАС ВЫ ЗАБУДЕТЕ ВСЁ, ЧТО БЫЛО РАНЬШЕ...");

Thread.Sleep(5000);

// Делегат для типа Timer

TimerCallback a = new TimerCallback(numbers.ForTimer);

Timer timer = new Timer(a, null, 0, 1000);

WriteLine("Нажмите чтоб выйти");

ReadLine();

#endregion

}

}

}

**Лабораторная 16. Платформа параллельных вычислений.**

using System;

using System.Collections.Concurrent;

using System.Diagnostics;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using System.Runtime.CompilerServices;

using static System.Console;

namespace Lab16

{

public partial class Program

{

private static void Main(string[] args)

{

#region [ The 1st task ]

WriteLine("\t\t\tЗадание 1");

Action action1 = new Action(SieveOfEratosthenes);

Stopwatch watch = Stopwatch.StartNew();

task1 = Task.Factory.StartNew(action1);

task1.Wait();

task1.Dispose();

watch.Stop();

WriteLine("Таск выполнен: " + task1.IsCompleted.ToString());

WriteLine("Статус: " + task1.Status.ToString());

WriteLine("Время выполнения для таска: " + watch.Elapsed);

#endregion

#region [ The 2nd task ]

WriteLine("\n\t\t\tЗадание 2");

CancellationTokenSource token = new CancellationTokenSource();

(task1 = new Task(action1, token.Token)).Start();

token.Cancel();

WriteLine("Токен отмены сработал!");

#endregion

#region [ The 3rd task ]

WriteLine("\n\t\t\tЗадание 3");

Task<int> rand1 = new Task<int>(RandomFor10),

rand2 = new Task<int>(RandomFor20),

rand3 = new Task<int>(RandomFor30);

rand1.Start();

WriteLine("First Value: " + rand1.Result);

rand2.Start();

WriteLine("Second Value: " + rand2.Result);

rand3.Start();

WriteLine("Third Value: " + rand3.Result);

Task<int> avg = new Task<int>(() => AverageValue(rand1.Result, rand2.Result, rand3.Result));

avg.Start();

WriteLine("Average: " + avg.Result);

#endregion

#region [ The 4th task ]

WriteLine("\n\t\t\tЗадание 4");

WriteLine("Таск продолжения: ");

Task<int> contTask1 = new Task<int>(RandomFor10);

// Планировка на основе завершения множества предшествующих задач

Task<int> contTask2 = contTask1.ContinueWith((task) => RandomFor20());

Task<int> contTask3 = contTask2.ContinueWith((task) => RandomFor30());

Task<int> contTask4 = contTask3.ContinueWith((task) => AverageValue(contTask1.Result, contTask2.Result, contTask3.Result));

contTask1.Start();

WriteLine("Первое значение: " + contTask1.Result);

WriteLine("Второе значение: " + contTask2.Result);

WriteLine("Третье значение: " + contTask3.Result);

WriteLine("Среднее значение: " + contTask4.Result);

contTask1.Dispose();

contTask2.Dispose();

contTask3.Dispose();

contTask4.Dispose();

WriteLine("Ожидающий");

Task<int> wait = Task.Run(() => Enumerable.Range(1, 100000).Count(n => (n % 2 == 0)));

TaskAwaiter<int> awaiter = wait.GetAwaiter();

awaiter.OnCompleted(() => { int res = awaiter.GetResult(); WriteLine(res); });

#endregion

#region [ The 5th task ]

WriteLine("\n\t\t\tЗадание 5");

Stopwatch stW = new Stopwatch();

int[] arr1 = new int[10000000];

int[] arr2 = new int[10000000];

Random random = new Random();

stW.Restart();

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

{

arr1[i] = random.Next(0, 300);

arr2[i] = random.Next(0, 300);

}

stW.Stop();

WriteLine("For: " + stW.Elapsed);

stW.Restart();

Parallel.For(0, arr1.Length, i =>

{

arr1[i] = random.Next(0, 300);

arr2[i] = random.Next(0, 300);

});

stW.Stop();

WriteLine("Параллельный for: " + stW.Elapsed);

stW.Restart();

Parallel.ForEach<int>(arr1, i =>

{

arr1[i] = random.Next(0, 300);

arr2[i] = random.Next(0, 300);

});

stW.Stop();

WriteLine("Параллельный foreach: " + stW.Elapsed);

#endregion

#region [ The 6th task ]

WriteLine("\n\t\t\tЗадание 6");

Parallel.Invoke(() =>

{

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

{

WriteLine("Параллельное выполнение блока 1 - " + i);

}

WriteLine("1 готов!");

},

() =>

{

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

{

WriteLine("Параллельное выполнение 2 - " + i);

}

WriteLine("2 готов!");

});

#endregion

#region [ The 7th task ]

WriteLine("\n\t\t\tЗадание 7");

MyBlock = new BlockingCollection<string>(5);

Task Shop = new Task(AddProduct);

Task Clients = new Task(PurchasedProduct);

Shop.Start();

Clients.Start();

Shop.Wait();

Clients.Wait();

#endregion

#region [ The 8th task ]

WriteLine("Задание 8");

Async();

string p = ReadLine();

WriteLine(p + p + p + "Нажмите, чтобы закончить работу...");

ReadKey();

#endregion

}

}

}

partial class Program

{

public static Task task1;

public static void SieveOfEratosthenes()

{

int n = 1000;

WriteLine("Текущий ID таска: " + Task.CurrentId.ToString());

var numbers = new List<uint>();

// Заполнение списка numbers числами от 2 до n-1

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

{

numbers.Add(i);

}

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

{

for (var j = 2u; j < n; j++)

{

numbers.Remove(numbers[i] \* j);

}

}

foreach (int a in numbers)

{

WriteLine(a);

}

}

}

partial class Program

{

public static int RandomFor10()

{

Random rndm = new Random();

return rndm.Next(10);

}

public static int RandomFor20()

{

Random rndm = new Random();

return rndm.Next(20);

}

public static int RandomFor30()

{

Random rndm = new Random();

return rndm.Next(30);

}

public static int AverageValue(int value1, int value2, int value3)

{

return (value1 + value2 + value3) / 3;

}

}

using System;

using System.Collections.Generic;

using System.Threading;

using System.Collections.Concurrent;

namespace Lab16

{

partial class Program

{

public static BlockingCollection<string> MyBlock;

public static void AddProduct()

{

Random r = new Random();

int x;

List<string> products = new List<string>() {"Ёлка", "Гирлянда", "Шоколадка", "Шампанское", "Мандарины" };

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

{

x = r.Next(0, products.Count - 1);

Console.WriteLine("Добавлен товар: " + products[x]);

MyBlock.Add(products[x]);

products.RemoveAt(x);

Thread.Sleep(r.Next(1, 3));

}

MyBlock.CompleteAdding();

}

public static void PurchasedProduct()

{

string str;

while (MyBlock.IsAddingCompleted == false)

{

if (MyBlock.TryTake(out str) == true)

Console.WriteLine("Был куплен товар: " + str);

if (MyBlock.TryTake(out str) != true)

Console.WriteLine("Покупатель ушел.");

}

}

}

}

using System;

using System.Threading.Tasks;

using System.Threading;

// Tpl – библиотека параллельных задач.

// Task –данный класс описывает отдельную задачу, которая запускается асинхронно

//в одном из потоков пула потоков.

namespace Lab16

{

partial class Program

{

public static void ForAsync()

{

for (int i = 0; i < 30; i++)

if (i % 3 == 0)

{

Console.Write(i + ", ");

Thread.Sleep(1000);

}

}

public static async void Async()

{

Console.WriteLine("Асинхронная функция:");

await Task.Run(() => ForAsync());

Console.WriteLine("Готово!");

}

}

}

**Контрольная работа.**

using System;

namespace Control

{

public enum PM\_AM : uint

{ PM = 1, AM = 0}

public class Time : ICheck

{

public uint h { get; set; }

public uint m { get; set; }

public uint s { get; set; }

public PM\_AM someTime;

// переопределение ToString

public override string ToString()

{

string ss = s.ToString();

string sm = m.ToString();

string sh = h.ToString();

string some = someTime.ToString();

string time = "Сейчас " + sh + ":" + sm + ":" + ss + " " + some;

return time;

}

public static bool operator >(Time a, Time b)

{

if (a.m > b.m || ((a.m == b.m) && (a.s > b.s)))

return true;

else return false;

}

public static bool operator <(Time a, Time b)

{

if (b > a) return true;

else return false;

}

public Time(uint h, uint m, uint s)

{

this.h = h;

this.m = m;

this.s = s;

}

public Time() { someTime = new PM\_AM(); }

public void Check()

{

if (h >= 21 && h <= 24 || h>=0 && h<=4)

{

Console.WriteLine("Ночь");

}

if (h >= 17 && h <= 21)

{

Console.WriteLine("Вечер");

}

if (h >= 10 && h <= 17)

{

Console.WriteLine("День");

}

if (h >= 4 && h <= 10)

{

Console.WriteLine("Утро");

}

}

}

}

using System;

namespace Control

{

interface ICheck

{

void Check();

}

public class FullTime : Time, ICheck

{

public FullTime() : base()

{

}

public new void Check()

{

uint vh = 24 - h;

uint vmin = 60 - m;

uint vsec = 60 - s;

Console.WriteLine($"Осталось до полуночи: {vh}:{vmin}:{vsec}");

}

}

}

using System;

namespace Control

{

class Program

{

static void Main(string[] args)

{

// 1

Console.WriteLine((int)char.MaxValue);

string str = "-885";

int inegerVal = Int32.Parse(str);

Console.WriteLine(inegerVal + "\n");

// 2

Time myTime = new Time(5,43,12);

myTime.someTime = (PM\_AM)1;

Console.WriteLine(myTime.ToString()+"\n");

Time myTime1 = new Time();

myTime1.h = 10;

myTime1.m = 10;

myTime1.s = 10;

myTime1.someTime = (PM\_AM)1;

Console.WriteLine(myTime1.ToString() + "\n");

Time myTime2 = new Time(7, 43, 12);

var test = myTime > myTime2;

Console.WriteLine(test + "\n");

// 3

Time test1 = new Time();

test1.h = 16;

test1.m = 20;

test1.s = 45;

test1.someTime = (PM\_AM)1;

Console.WriteLine(test1.ToString());

FullTime test2 = new FullTime();

test2.h = 6;

test2.m = 7;

test2.s = 42;

test2.someTime = (PM\_AM)0;

Console.WriteLine(test2.ToString() + "\n");

test1.Check();

test2.Check();

}

}

}