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

using System;

using System.Text; //для класса StringBuilder

//~~~~~ класс Convert

Console.Write("Enter your name : ");

string name = Console.ReadLine();

Console.Write("Age : ");

int age = Convert.ToByte(Console.ReadLine());

Console.WriteLine($"Name : {name}, age : {age}");

//явная упаковка/распаковка

int a = 10; //данные переменной размещаются в стеке

object o\_a = a; //упаковка (object - базовый класс для всех типов данных в c#, т.к. object - класс, то его данные размечаются в куче)

//таким образом в переменной o\_a получаем ссылку на данные в куче

int b = (int)o\_a; //распаковка

//неявная упаковка/распаковка

int c = 1;

c.GetType(); // возвращает тип данных переменной c

Console.Write(c + 8);

//свойства Nullable-типа

int? x = null;

Console.WriteLine($"x.HasValue : {x.HasValue}"); //x = null, значит выведет False

Console.WriteLine($"x ?? \"-1\" : {x ?? -1}"); // если x - null, выведет -1, иначе значение x

//Console.WriteLine($"x.Value : {x.Value}"); //приведет к ошибке, так как x = null

Console.WriteLine($"x : {x}"); //выведет пустоту

//математические операции с Nullable-типом

x = 2;

int? y = null;

Console.WriteLine($"\nx = 2, y = null \n" +

$"x + 3 = {x + 3} \n" + //5

$"x < 1 : {x < 1} \n" + //False

$"x + y = {x + y}"); //number "+,-,/,\*" null = null

//явное приведение Nullable-типа

float f\_a = (float)x; //если Nullable-тип имеет значение отличное от null, то мы можем преобразовать его к другому типу по правиласм преобразования

Console.WriteLine("\n" + f\_a);

string st1 = "Ito";

string st2 = "Mitsu";

//Для сравнения строк применяется статический метод Compare

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

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

Console.WriteLine(String.Compare(st1, st2));

string s1 = "Broken boy, ";

string s2 = "How does ";

string s3 = "it feel?";

string concat\_plus = s1 + s2 + s3;

string concat\_method = String.Concat(s1, s2, s3);

string concat\_join = String.Join("", s1, s2, s3);

StringBuilder str2d = new StringBuilder("Can you please sit the fuck down"); //выделяет больше памяти, чем нужно в данный момент, для возможности менять строку

int[,] nums = { { 0, 1, 2, 3 }, { 4, 5, 6, 7 }, { 8, 9, 10, 11 } };

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

int cols = nums.GetUpperBound(1) + 1;

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

{

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

{

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

}

Console.WriteLine("\n");

}

float[][] arr = new float[3][];

arr[0] = new float[2];

arr[1] = new float[3];

arr[2] = new float[4];

(int, string, char, string, ulong) cor = (7, "Kyou kara Ore wa!!", 'A', "Rainforest Alliance", 45);

Console.WriteLine(cor);

Console.WriteLine($"\n1 cortege element : {cor.Item1}");

Console.WriteLine($"2 cortege element : {cor.Item2}");

Console.WriteLine($"3 cortege element : {cor.Item3}");

Console.WriteLine($"4 cortege element : {cor.Item4}");

Console.WriteLine($"5 cortege element : {cor.Item5}");

//деконструирование кортежа

//явное объявление первых двух переменных и ключевое слово var, чтобы C# сам определил тип третьей переменной

(string song, int num, var fl) = ("Smells Like Teen Spirit", 10, 9.4f);

//C# объявлет тип всех переменных

var (int\_num, cha) = (-10, 'E');

//вы можете разложить кортеж на уже объявленные переменные

int a4c;

int b4c = 9;

(a4c, b4c) = (10, 7);

(int, int, int, int, int) cor1 = (10, 20, 9, 0, 40);

(int, int, int, int, int) cor2 = (20, 10, 9, 75, -2);

//позволяет сравнивать кортежи с разным количеством элементов

Console.WriteLine("cor2.Equals(cor1) : " + cor2.Equals(cor1)); //False

//компилируются только, если количество элементов в первом кортеже равно количеству во втором

Console.WriteLine("cor1 == cor2 : " + (cor1 == cor2)); //False

Console.WriteLine("cor2.CompareTo(cor1) : " + cor2.CompareTo(cor1)); //1

Console.WriteLine("cor1.CompareTo(cor2) : " + cor1.CompareTo(cor2)); //-1

void ch(int a)

{

try

{

checked

{

a = a + 8;

}

Console.WriteLine(a);

}

catch (OverflowException)

{

Console.WriteLine("Переполнение");

}

}

void unch(int a)

{

unchecked

{

a = a + 8;

}

Console.WriteLine(a);

}

int max\_int = 2147483647;

ch(max\_int);

unch(max\_int);

**№ 3 Проектирование типов. Классы.**

//Закрытый конструктор (с модификатором private) не позволяет создавать определенного вида объекты

//То есть, например наш конструктор не принимает никаких параметров => мы не сможем создать экземпляр класса

//без передачи параметров

private Student() { }

//поле-константа (минимальный средний балл для получения стипендии)

public const float min\_scolarship\_score = 5.0f;

//---------------------------------------

//поле - только для чтения

public readonly Guid id; //Guid - спец. структура для создания уникальных id

//сравнение объектов

public override bool Equals(object obj)

{

//типы сравниваемых объектов должны быть одинаковыми

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

return false;

//obj приводим к виду объекта типа Student

Student st = (Student)obj;

return (

this.surname == st.surname &&

this.bday\_date == st.bday\_date &&

this.faculty == st.faculty

);

}

//расчет возраста студента

public int age()

{

var today = DateTime.Today;

var age = today.Year - this.bday\_date.Year;

//отнимаем полученный возраст от сегодняшней даты

//если в этом году у студента ещё не было дня рождения, то результат будет меньше даты дня рождения студента

//так как age - это возраст, которого достигнет человек в этом году, то отнимаем 1 для получения текущего возраста

if (this.bday\_date.Date > today.AddYears(-age)) age--;

return age;

}

**№ 4 Перегрузка операций, методы расширения и вложенные типы**

Класс – множество **Set**. Дополнительно перегрузить следующие операции: > проверка на принадлежность (типа операции in множественного типа)\*  пересечение множеств; < проверка на подмножество; явный Date()приведение типа

Методы расширения:

1) Выделение первого числа, содержащегося в строке

2) Удаление положительных элементов из множества

using System;

using System.Collections.Generic; //List<type\_name>

//Проверка на подмножество

//Множество subset принадлежит множеству set (является его подмножеством)

//Если любой элемент множества subset также принадлежит множеству set

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

{

int subset\_length = subset.data.Count;

int num\_of\_subset\_elements\_in\_set = 0;

//если в множестве data содержится число num, значит num принадлежит data

foreach (int subset\_item in subset.data)

{

if (set.data.Contains(subset\_item)) {

num\_of\_subset\_elements\_in\_set++;

}

}

if (num\_of\_subset\_elements\_in\_set == subset\_length)

{

return true;

}

return false;

}

Console.WriteLine($"set3 является подмножеством set2 (set2 < set3) : {set2 < set3}");

public class Set

{

//вложенный класс Date(дата создания)

public class Date

{

public readonly DateTime dateOfCreation = DateTime.Now;

}

public Date dateObj = new Date();

}

using System;

using System.Collections.Generic;

using static Laba4.Program; //для Set !!!

namespace Laba4

{

static class Extension\_methods

{

//выделение первого числа в строке

public static int FindFirstNum(this string str)

{

string num = "";

foreach(char i in str)

{

//если символ строки равен от 0 до 9, значит записываем его в строку num

if (i >= 48 && i <= 57)

{

num += i;

}

//если в строке num уже есть число, а мы встретили символ, отличный от цифры - значит мы уже выделили первое число

else if (num != "")

{

break;

}

}

//если в строке чисел нет

if (num == "")

{

return -1;

}

else

{

return Convert.ToInt32(num);

}

}

//------------------------------------------------------------------

//Удаление положительных элементов из множества Set

public static List<int> RemovePosEls(this Set set)

{

List<int> posSet = new List<int> { };

foreach(int i in set.Data)

{

if (i < 0)

{

posSet.Add(i);

}

}

return posSet;

}

}

}

**№ 5 Наследование, полиморфизм, абстрактные классы и интерфейсы**

//абстрактный класс

abstract class Person

{

public Person(string name, string surname)

{

Name = name;

Surname = surname;

}

public string Name { get; set; }

public string Surname { get; set; }

//абстрактный метод

public abstract void Print();

}

//класс-наследник абстрактного класса Person

class Director : Person

{

public Director(string name, string surname) : base(name, surname) { }

//обязательное переопределение абстрактного метода родительского класса

public override void Print()

{

Console.WriteLine("Имя режиссёра : " + this.Name + ", фамилия : " + this.Surname);

}

public void SaySmth()

{

Console.WriteLine("Я режиссёр!");

}

}

//интерфейс для вывода описания произведения

interface IDescription

{

void PrintDescription();

}

//------------------------------------------------------

//абстрактный класс к заданию 4, который содержит метод с тем же названием, что и метод интерфейса IDescription

abstract class DescriptionClass

{

public abstract void PrintDescription();

}

partial class Advertising : DescriptionClass, IDescription

{

//конструктор

public Advertising(string title)

{

Title = title;

}

//методы

public string Title { get; set; }

public string CustomerCompany { get; set; }

public string AdvertisingAgency { get; set; }

public float DurationMinutes { get; set; }

public DateTime PremiereDate { get; set; }

public string Description { get; set; }

//явная реализация метода интерфейса IDescription

void IDescription.PrintDescription()

{

Console.WriteLine($"(интерфейс) Описание \"{Title}\" : {Description}");

}

public override void PrintDescription()

{

Console.WriteLine($"(абстрактный класс) Описание \"{Title}\" : {Description}");

}

}

//задание 4, вызов одноименного метода

Advertising adv1 = new Advertising("adv1");

adv1.PrintDescription(); //вызовется переопределение абстрактного метода

((IDescription)adv1).PrintDescription(); //вызовется реализация метода интерфейса

//Создаем объект типа Director, ссылку на объект передаем в переменную типа Person

//Тип Person является абстрактным классом, от которого Director наследуется

//Таким образом у нас ссылка типа Person, которая хранит данные типа Director

Person P\_NarDir = new Director("Наруто", "Узумаки");

//Получение данных

//P\_NarDir.SaySmth(); //класс Director содержит метод SaySmth, ссылка типа Person не имеет доступа к этому методу (ошибка на этапе компиляции)

//явное преобразование

Director D\_NarDir = P\_NarDir as Director;

//Получение данных

D\_NarDir.SaySmth(); //таким образом тип ссылки ограничивает область доступных данных и методов

if (D\_NarDir != null)

{

//Переменная, типа Director, записанная по ссылке Person, может быть записана по ссылке типа Director при явном преобразовании к этому типу

Console.WriteLine("\n+ Преобразование ссылки типа абстрактного класса в ссылку типа класса, " +

"объект которого хранила ссылка типа абстрактного класса, удалось!");

}

else

{

Console.WriteLine("\n- Преобразование ссылки типа абстрактного класса в ссылку типа класса, " +

"объект которого хранила ссылка типа абстрактного класса, НЕ удалось!");

}

bool P\_NarDir\_is\_Dir = P\_NarDir is Director;

if (P\_NarDir\_is\_Dir)

{

Console.WriteLine("\nPerson P\_NarDir = new Director(\"Наруто\", \"Узумаки\"); =>" +

" P\_NarDir типа Director, Person - абстрактный класс");

}

else

{

Console.WriteLine("\nPerson P\_NarDir = new Director(\"Наруто\", \"Узумаки\"); =>" +

" P\_NarDir НЕ типа Director, Person - абстрактный класс");

}

**№ 6 Структуры, перечисления, классы контейнеры и контроллеры**

namespace Laba6

{

public enum WaysToCreateACartoon {

пластилиновый,

рисованный,

кукольный,

компьютерный,

художественный,

}

struct Director

{

public string Name { get; set; }

public string Surname { get; set; }

public Director(string name, string surname)

{

Name = name;

Surname = surname;

}

}

}

**№ 7 Исключения**

using System;

namespace Laba7

{

class Exceptions

{

//проверка даты премьеры

public class PremiereDate\_Exception : Exception

{

public PremiereDate\_Exception(string message) : base(message) { }

}

//если в качестве названия передана строка нулевой длины или null будет выбрасываться исключение

public class Title\_Exception : ArgumentException

{

public Title\_Exception(string message) : base(message) { }

}

//проверка продолжительности

public class DurationMinutes\_Exception : ArgumentException

{

public DurationMinutes\_Exception(string message) : base(message) { }

}

}

}

class TVProgram

{

private string \_title;

public string Title {

get {

return \_title;

}

set {

if (value == null)

{

throw new Exceptions.Title\_Exception("Необходимо задать название!");

}

else if (value.Length == 0)

{

throw new Exceptions.Title\_Exception("Пустая строка не может быть названием.");

}

else

{

\_title = value;

}

}

}

**}**

**№ 8 Обобщения**

//значение по умолчанию

class Account<T>

{

    T id = default(T);

}

// в качестве ограничений можно исп. классы, интерфейсы,

// class => универсальный параметр должен представлять класс

// struct => универсальный параметр должен представлять структуру

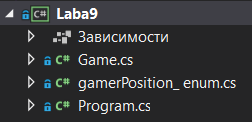
// new() => универсальный параметр должен представлять тип, который имеет общедоступный (public) конструктор без параметров

// При этом использовать в качестве ограничения конкретные структуры в отличие от классов нельзя.

class Account<T> where T : struct

{ }

**№ 9 Делегаты, события и лямбда выражения**

****

**Game.cs**

using System;

namespace Laba9

{

class Game

{

public delegate void Attack(int energyBeforeHit, int expendedEnergy, int Energy, bool edible);

public delegate void Heal(int energyBeforeHeal, int expendedEnergy, int Energy);

public Game(gamerPosition position)

{

//каждому игроку вначале игры дается энергия = 100 единиц

Energy = 100;

Position = position;

}

public gamerPosition Position { get; set; }

public event Attack AttackEvent; //событие: игрок атаковал

public event Heal HealEvent; //событие: игрок вылечил

//энергия игрока

private int energy;

public int Energy

{

get

{

return energy;

}

private set

{

energy = value;

}

}

//игрок нанес удар

public void Hit(bool edible)

{

//записываем значение энергии до удара

int energyBeforeHit = Energy;

//затраченная энергия (зависит от позиции игрока)

int expendedEnergy = 0;

//в зависимости от того, является ли игрок воином или целителем, на аттаку он затратит разное количество энергии

switch (this.Position)

{

case gamerPosition.Warrior:

if (edible) expendedEnergy = 5;

else expendedEnergy = 3;

break;

case gamerPosition.Healer:

if (edible) expendedEnergy = 7;

else expendedEnergy = 5;

break;

}

if (energyBeforeHit > expendedEnergy)

{

//отнимаем от значения энергии до удара количество затраченной на этот удар энергии

Energy -= expendedEnergy;

//если игрок аттаковал нечто съестное, то он это съедает и получает энергию

if (edible)

{

Energy += 10;

}

}

//извещаем подписчиков на событие AttackEvent, о том что игрок атаковал

AttackEvent?.Invoke(energyBeforeHit, expendedEnergy, Energy, edible);

}

//игрок кого-то вылечил

public void Help()

{

int energyBeforeHeal = Energy;

int expendedEnergy = 0;

switch (this.Position)

{

case gamerPosition.Warrior:

expendedEnergy = 20;

break;

case gamerPosition.Healer:

expendedEnergy = 5;

break;

}

if (energyBeforeHeal > expendedEnergy)

{

Energy -= expendedEnergy;

}

HealEvent?.Invoke(energyBeforeHeal, expendedEnergy, Energy);

}

public void getEnergy()

{

Energy += 10;

Console.WriteLine("\*\*Вы пополнили запас энергии на 10 единиц\*\*");

}

}

}

**gamerPosition\_enum.cs**

namespace Laba9

{

//игрок может быть воином или целителем

enum gamerPosition

{

Warrior,

Healer

}

}

**Program.cs**

using System;

namespace Laba9

{

class Program

{

delegate void getCurrentState();

static void Main(string[] args)

{

Game gamer = null;

byte pos; //позиция игрока

byte action; //действие игрока

while (true)

{

Console.WriteLine("--> Выберите игрока <--\n1 - воин \n2 - целитель \n3 - выход");

byte.TryParse(Console.ReadLine(), out pos);

if(pos == 3)

{

return;

}

else if(pos == 1 || pos == 2)

{

break;

}

else

{

Console.WriteLine("Игрок не зарегестрирован. \nВыберите один из предложенных вариантов.");

}

}

switch (pos)

{

case 1:

gamer = new Game(gamerPosition.Warrior);

break;

case 2:

gamer = new Game(gamerPosition.Healer);

break;

}

gamer.AttackEvent += StateAfterAttack;

gamer.HealEvent += StateAfterHeal;

//Лямбда

getCurrentState displayCurrentEnergy = () => Console.WriteLine($"\*\*Ваша энергия = {gamer.Energy}\*\*");

while (true)

{

Console.WriteLine("\n--> Что вы хотите сделать? <--");

Console.WriteLine("1 - аттаковать и кушать \n2 - просто аттаковать \n3 - вылечить другого игрока " +

"\n4 - выполнить задание и пополнить запас энергии \n5 - посмотреть текущий уровень энергии \n6 - выход");

bool suitableValue = byte.TryParse(Console.ReadLine(), out action);

if (!suitableValue)

{

Console.WriteLine("Выберите один из предложенных вариантов.");

continue;

}

switch (action)

{

case 1:

gamer.Hit(true);

break;

case 2:

gamer.Hit(false);

break;

case 3:

gamer.Help();

break;

case 4:

gamer.getEnergy();

break;

case 5:

displayCurrentEnergy();

break;

case 6:

return;

default:

Console.WriteLine("Выберите один из предложенных вариантов.");

break;

}

}

}

static public void StateAfterAttack(int energyBeforeHit, int expendedEnergy, int Energy, bool edible)

{

. . .

}

}

}

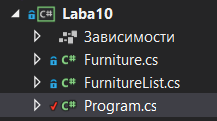
//Делегат Action является обобщенным, принимает параметры и возвращает значение void:

public delegate void Action<T>(T obj);

// Func<T> позволяет вызывать методы с типом возврата и разными типами параметров

Func<int, int, string> op; //возвращаемый тип – string

**№ 10 Коллекции**



**Furniture.cs**

namespace Laba10

{

class Furniture

{

//--> свойства мебели

public string Name { get; set; }

public int Price { get; set; }

public Furniture(string name, int price)

{

Name = name;

Price = price;

}

}

}

**FurnitureList.cs**

using System;

using System.Collections.Generic;

using System.Collections;

namespace Laba10

{

//реализация интерфейса IList для создания коллекции типа FurnitureList для хранения объектов типа Furniture

class FurnitureList : IList<Furniture>

{

//коллекция объектов объектов

public ArrayList fList = new ArrayList();

//метод для вывода коллекции FurnitureList

public void Print()

{

foreach (var item in fList)

{

Console.WriteLine(item.ToString());

}

}

//реализация интерфейса IList

public Furniture this[int index]

{

get

{

if ((index >= 0) && (index < fList.Count))

{

return (Furniture)fList[index];

}

else

{

Console.WriteLine("-- > Вы вышли за пределы размера коллекции < --");

Console.WriteLine("-- > Поэтому будет выведен первый элемент коллекции < --");

return (Furniture)fList[0];

}

}

set { throw new NotImplementedException(); }

}

public int Count => fList.Count;

public bool IsReadOnly { get; }

public void Add(Furniture item)

{

fList.Add(item);

}

public void Clear()

{

fList.Clear();

}

public bool Contains(Furniture obj)

{

foreach (var el in this)

{

if (el.Equals(obj))

return true;

}

return false;

}

public void CopyTo(Furniture[] array, int arrayIndex)

{

fList.CopyTo(array, arrayIndex);

}

public int IndexOf(Furniture item)

{

return fList.IndexOf(item);

}

public void Insert(int index, Furniture item)

{

fList.Insert(index, item);

}

public bool Remove(Furniture item)

{

fList.Remove(item);

return true;

}

public void RemoveAt(int index)

{

try

{

fList.RemoveAt(index);

}

catch

{

Console.WriteLine("--> Вы вышли за пределы размера коллекции <--");

}

}

public IEnumerator<Furniture> GetEnumerator()

{

foreach (var item in fList)

{

yield return (Furniture)item;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Collections.Specialized;

//класс FurnitureList представляет собой коллекцию для работы с объектами класса Furniture,

//в FurnitureList определена переменная fList типа коллекции ArrayList, для хранения объектов Furniture

namespace Laba10

{

class Program

{

static void Main(string[] args)

{

//-------------- 1 ЗАДАНИЕ --------------

FurnitureList fur = new FurnitureList();

fur.Add(new Furniture("Подушка \"Junior Green\"", 227));

fur.Add(new Furniture("Подушка \"Ocean Fusion L\"", 430));

//-------------- 2 ЗАДАНИЕ --------------

List<int> nums = new List<int> { 1, 2, 3, 4, 5, 6 };

//b. Удалите из коллекции n последовательных элементов

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

Console.WriteLine("С какого индекса нужно удалить элементы : ");

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

if (index >= 0 && index < nums.Count - 1)

{

Console.WriteLine("Сколько элементов нужно удалить?");

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

if (n > 0)

{

if (index + n > nums.Count)

n = nums.Count - index;

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

{

nums.RemoveAt(index);

Console.WriteLine(i);

}

}

}

else

{

Console.WriteLine("-- > Вы вышли за пределы размера коллекции < --");

}

//c. Добавьте другие элементы (используйте все возможные методы добавления для вашего типа коллекции).

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

nums.Add(2); // добавление элемента

nums.AddRange(new int[] { 7, 8, 9, 10 }); //добавление в список массива

nums.Insert(1, 99);// вставляем на второе место в списке число 99

//d. Создайте вторую коллекцию (из таблицы выберите другой тип коллекции) и заполните ее данными из первой коллекции.

Dictionary<char, int> dic = new Dictionary<char, int>();

for (int i = 0, k = 97; i < nums.Count; i++, k++)

{

char key = (char)k;

dic.Add(key, nums[i]);

}

foreach(char c in dic.Keys)

{

Console.WriteLine(c + " : " + dic[c]);

}

//Объект наблюдаемой коллекции ObservableCollection<T>

var furObserve = new ObservableCollection<Furniture>

{

fur[0],

fur[1],

};

Console.WriteLine("\n\nИзначально в наблюдаемой коллекции 2 объекта");

foreach (var item in furObserve)

{

Console.WriteLine(item);

}

furObserve.CollectionChanged += FurnitureList\_CollectionChanged;

Console.WriteLine("\n--> Начинаем изменять коллекцию <--");

furObserve.Add(fur[2]);

furObserve.Remove(fur[2]);

}

public static void FurnitureList\_CollectionChanged(object sender,

NotifyCollectionChangedEventArgs e)

{

switch (e.Action)

{

case NotifyCollectionChangedAction.Add:

Furniture newFur = e.NewItems[0] as Furniture;

Console.WriteLine($"Добавлен новый объект : {newFur.ToString()}");

break;

case NotifyCollectionChangedAction.Remove:

Furniture oldFur = e.OldItems[0] as Furniture;

Console.WriteLine($"Удален объект : {oldFur.ToString()}");

break;

case NotifyCollectionChangedAction.Replace:

Furniture replacedFur = e.OldItems[0] as Furniture;

Furniture replacingFur = e.NewItems[0] as Furniture;

Console.WriteLine($"Объект {replacedFur.ToString()} заменен объектом {replacingFur.ToString()}");

break;

}

}

}

}

**№ 11 LINQ to Object**

using System;

using System.Collections.Generic;

using System.Linq;

namespace Laba11

{

class Program

{

static void Main(string[] args)

{

//1 ЗАДАНИЕ

//Задайте массив типа string, содержащий 12 месяцев(June, July, May, December, January ….).

//Используя LINQ to Object напишите запрос выбирающий последовательность месяцев с длиной

//строки равной n, запрос возвращающий только летние и зимние месяцы, запрос вывода месяцев

//в алфавитном порядке, запрос считающий месяцы содержащие букву «u» и длиной имени не менее 4 - х..

string[] months = { "January", "February", "March", "April", "May",

"June", "July", "August", "September", "October", "November", "December" };

foreach (string month in months)

{

Console.Write(month + " ");

}

//

Console.Write("\n\n-->Введите какой длины названия месяцев нужно выбрать : ");

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

var monthsLen = from month in months

where month.Length == n

select month;

foreach(string month in monthsLen)

{

Console.Write(month + " ");

}

//

Console.WriteLine("\n\n-->Выборка из только зимних и летних месяцев последовательности");

string[] sumWintMonths\_arr = { "January", "February", "June", "July", "August", "December" };

var sumWintMonths\_request = months.Intersect(sumWintMonths\_arr);

foreach (string month in sumWintMonths\_request)

{

Console.Write(month + " ");

}

//

//Каждый раз, когда в выборке появлется буква 'u', название которому она принадлежит входит в выборку

//Так как в названии может быть несколько 'u', то один и тот же месяц может попасть в выборку несколько раз

//Поэтому используем метод Distinct(), который убирает дублирующиеся элементы

int amountMonthsWithLet\_u = (from month in months

from letter in month

where letter == 'u' && month.Length >= 4

select month).Distinct().Count();

Console.WriteLine("\n\n-->Количество названий месяцев с буквой \"u\" = " + amountMonthsWithLet\_u);

//

var sorted = from month in months

orderby month

select month;

Console.WriteLine("\n-->Список, отсортированный по алфавиту");

foreach (string month in sorted)

{

Console.Write(month + " ");

}

//2 ЗАДАНИЕ

//Создайте коллекцию List<T> и параметризируйте ее типом(классом) из лабораторной №3

//(при необходимости реализуйте нужные интерфейсы).Заполните ее минимум 8 элементами.

Student st1 = new Student("Иван", "Хмельной", new DateTime(2000, 10, 23), "ФИТ", "ПОИБМС", 7);

Student st2 = new Student("Иван", "Сырок", new DateTime(2006, 9, 22), "ФИТ", "ПОИБМС", 7);

Student st3 = new Student("Ито", "Шинджи", new DateTime(1988, 12, 22), "ФИТ", "ПОИБМС", 7);

Student st4 = new Student("Мицухаши", "Такаши", new DateTime(1988, 12, 22), "ФИТ", "ПОИБМС", 7);

Student st5 = new Student("Маки", "Зенин", new DateTime(2000, 7, 26), "ФИТ", "ДЭиВИ", 10);

Student st6 = new Student("Мегуми", "Фушигуро", new DateTime(2001, 1, 2), "ФИТ", "ДЭиВИ", 9);

Student st7 = new Student("Тодзи", "Фушигуро", new DateTime(1980, 6, 8), "ФИТ", "ПОИБМС", 7);

Student st8 = new Student("Юдзи", "Итадори", new DateTime(2002, 12, 18), "ФИТ", "ДЭиВИ", 9);

Student st9 = new Student("Сугуру", "Гето", new DateTime(1990, 6, 28), "ФИТ", "ПОИТ", 5);

List<Student> students = new List<Student>() { st1, st2, st3, st4, st5, st6, st7, st8, st9 };

//3 ЗАДАНИЕ

//На основе LINQ сформируйте запросы по вариантам

Console.WriteLine("\n\n--> Студенты спецальности \"ПОИБМС\", отсортированные по имени и фамилии");

string spec = "ПОИБМС";

var specOrdered = from student in students

where student.speciality == spec

orderby student.name, student.surname

select student;

foreach (var student in specOrdered)

{

Console.WriteLine(student.ToString());

}

//

Console.WriteLine("--> Cписок 6 учебной группы, факультета \"ПИМ\"");

string fac = "ПИМ";

int gr6 = 6;

var facGr = from student in students

where student.faculty == fac && student.Group == gr6

select student;

foreach (var student in facGr)

{

Console.WriteLine(student.ToString());

}

//

Console.WriteLine("--> Вывод информации о самом молодом студенте");

var youngest\_stud\_bday\_date = students.Max(student => student.bday\_date);

var youngest\_stud = from student in students

where student.bday\_date == youngest\_stud\_bday\_date

select student;

foreach (var student in youngest\_stud)

{

Console.WriteLine(student.ToString());

}

//

Console.Write("--> Количество студентов 7 группы : ");

int gr7 = 7;

var amountStudGr = (from student in students

where student.Group == gr7

select student).Count();

var amountStudGr\_exMeth = students.Where(student => student.Group == gr7).Count();

Console.Write(amountStudGr);

Console.WriteLine(" (запрос Linq), " + amountStudGr\_exMeth + " (метод расширения)");

//

Console.WriteLine("\n--> Вывод первого студента с именем \"Иван\"");

string name = "Иван";

var studName = (from student in students

where student.name == name

select student).FirstOrDefault();

if (studName != null)

Console.WriteLine(studName);

else

Console.WriteLine("Ничего не найдено");

//4 ЗАДАНИЕ

//Придумайте и напишите свой собственный запрос, в котором было бы не менее 5 операторов из разных категорий:

//условия, проекций, упорядочивания, группировки, агрегирования, кванторов и разбиения.

//Выбрать студентов старше 20, сгруппировать их по группам, затем создать новый объект с полями:

//groupNum - номер группы, Count - количество студентов в группе, studs - массив студентов данной группы

Console.WriteLine("\n--> Студенты старше 20, разбитые по своим группам\n");

var myReq = from student in students

where student.age() > 20

group student by student.Group into gr

select new

{

groupNum = gr.Key,

Count = gr.Count(),

studs = from st in gr

select st

};

foreach(var g in myReq)

{

Console.WriteLine($"{g.groupNum} группа , число студентов : {g.Count}");

foreach(Student st in g.studs)

{

Console.WriteLine(st.ToString());

}

}

//5 ЗАДАНИЕ

//Придумайте запрос с оператором Join

var characters = new List<Character>(){

new Character("Киллуа Золдик","Хантер x Хантер"),

new Character("Курапика","Хантер x Хантер"),

new Character("Эдвард Элрик","Стальной алхимик"),

new Character("Грид","Стальной алхимик"),

new Character("Чосо","Магическая битва"),

new Character("Куроро Люцифер","Хантер x Хантер"),

new Character("Гон Фрикс","Хантер x Хантер"),

new Character("Кугисаки Нобара","Магическая битва"),

};

var animes = new List<Anime>(){

new Anime("Магическая битва"),

new Anime("Хантер x Хантер"),

new Anime("Стальной алхимик"),

};

var result = characters.Join(animes,

c => c.anime,

a => a.name,

(c, a) => new { character = c.name, anime = a.name });

foreach(var item in result)

{

Console.WriteLine($"Аниме : {item.anime}, персонаж : {item.character}");

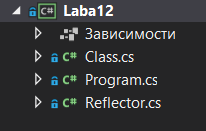
}

}

}

}

**№ 12 Рефлексия**

****

**Class.cs**

using System;

namespace Laba12

{

interface IPref

{

// отношение животного к чему-то

//(типа: любит сырую картошку => attitude - "любит", smth - "сырую картошку")

void pref(string attitude, string smth);

}

class Animal : IPref

{

public string Type { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public Animal(string type, string name, int age)

{

Type = type;

Name = name;

Age = age;

}

public Animal() { }

public void pref(string attitude, string smth)

{

Console.WriteLine($"{Type} {Name} {attitude} {smth}");

}

}

}

**Program.cs**

using System;

namespace Laba12

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("|||| Запись информации о типах в файлы ||||\n");

Reflector.WriteInfoToFile("Laba12.Animal", typeof(string), @"D:\ООП\_2к\_1с\Laba12\infoAnimal.txt");

Reflector.WriteInfoToFile("System.String", typeof(int), @"D:\ООП\_2к\_1с\Laba12\infoString.txt");

Console.WriteLine("\n|||| Создание экземпляра типа с помощью поиска " +

"наиболее подходящего конструктора (в соответствии с переданными параметрами) ||||\n");

Animal animal = Reflector.Create("Laba12.Animal", new object[] { "кролик", "Станислав", 2 })

as Laba12.Animal;

Console.WriteLine("Тип животного : " + animal.Type);

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

Console.WriteLine("Возраст : " + animal.Age);

Console.WriteLine("\n|||| Передача методу значений, считанных из файла ||||\n");

Animal dog = new Animal("собака", "Лика", 10);

Reflector.Invoke(dog, "Laba12.Animal", "pref", @"D:\ООП\_2к\_1с\Laba12\parameters.txt");

}

}

}

**Reflector.cs**

using System;

using System.Collections.Generic;

using System.Reflection;

using System.IO;

using System.Linq;

namespace Laba12

{

static class Reflector

{

// проверка существования указанного типа

static public void typeExist(string type)

{

//второй параметр отвечает за то, что произойдет, если указанного типа не будет найдено

//false - возврат null

//true - генерация исключения

Type typeInfo = Type.GetType(type, false);

//если указанного типа не было найдено, завершаем программу

if (typeInfo == null)

{

Console.WriteLine("Указанного вами типа не было найдено");

System.Environment.Exit(1);

}

}

// имя сборки, в которой определен класс

static public string assemblyName(string type)

{

typeExist(type);

Type t = Type.GetType(type);

string assemblyName = Assembly.GetAssembly(t).FullName;

return assemblyName;

}

// есть ли публичные конструкторы

static public bool containsPublConstructors(string type)

{

typeExist(type);

Type t = Type.GetType(type);

ConstructorInfo[] ctors = t.GetConstructors(BindingFlags.Public);

if (ctors == null)

return false;

else

return true;

}

//извлекает все публичные методы класса

static public MethodInfo[] getPublMethods(string type)

{

typeExist(type);

Type t = Type.GetType(type);

MethodInfo[] methods = t.GetMethods();

return methods;

}

static public FieldInfo[] getFields(string type)

{

typeExist(type);

Type t = Type.GetType(type);

FieldInfo[] fields = t.GetFields();

return fields;

}

static public PropertyInfo[] getProps(string type)

{

typeExist(type);

Type t = Type.GetType(type);

PropertyInfo[] props = t.GetProperties();

return props;

}

static public Type[] getInterfaces(string type)

{

typeExist(type);

Type t = Type.GetType(type);

Type[] interfaces = t.GetInterfaces();

return interfaces;

}

//выводит по имени класса имена методов, которые содержат заданный (пользователем) тип параметра

static public IEnumerable<MethodInfo> getMeth(string type, Type paramType)

{

typeExist(type);

Type t = Type.GetType(type);

MethodInfo[] methods = t.GetMethods();

var methWithDefineParam = methods // из массива methods

// берем каждый последующий элемент (meth) и получаем массив его параметров (meth.GetParameters)

.Where(meth => meth.GetParameters()

// у каждого последующего элемента в массиве пареметров (param) узнаем тип (param.ParameterType)

//и сравниваем с типом переданным методом getMeth (paramType)

.Where(param => param.ParameterType == paramType)

// если тип с подходящим параметром найден, то он попадает в итоговую выборку

.Count() != 0);

return methWithDefineParam;

}

// можно передать только методы, содержащие параметры типа string

// obj - объект типа type, для которого мы вызываем метод meth

// path - путь к файлу со значениями переметров метода

static public void Invoke(object obj, string type, string meth, string path)

{

typeExist(type);

Type t = Type.GetType(type);

MethodInfo method = t.GetMethod(meth);

int numOfParams = method.GetParameters().Length;

try

{

// значения параметров

object[] foundParamsValues = new object[numOfParams];

StreamReader fIn = new StreamReader(path);

string line; //строка считанная из файла

int i = -1;

do

{

i++;

line = fIn.ReadLine();

if (line != null)

foundParamsValues[i] = line;

} while (line != null && (i + 1) < numOfParams);

// динамический вызов метода

// первый параметр - объект, чей метод вызывается

// второй - массив значений параметров данного метода

method.Invoke(obj, foundParamsValues);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

// запись информации о типе в файл

static public void WriteInfoToFile(string type, Type paramType, string path)

{

typeExist(type);

string info = "Имя типа : " + type + "\n";

info += "Название сборки, в которой определен тип : " + assemblyName(type) + "\n";

info += "Содержит ли тип публичные конструкторы : " + containsPublConstructors(type) + "\n";

info += "Публичные методы :\n";

MethodInfo[] publMethods = getPublMethods(type);

foreach(var m in publMethods)

{

info += "--> " + m.Name + "\n";

}

info += "Поля :\n";

FieldInfo[] fields = getFields(type);

foreach (var f in fields)

{

info += "--> " + f.Name + "\n";

}

info += "Свойства :\n";

PropertyInfo[] props = getProps(type);

foreach (var p in props)

{

info += "--> " + p.Name + "\n";

}

info += "Реализуемые интерфейсы :\n";

Type[] interfaces = getInterfaces(type);

foreach (var i in interfaces)

{

info += "--> " + i.Name + "\n";

}

info += $"Методы с параметром типа {paramType} :\n";

IEnumerable<MethodInfo> methodsWithDefineParType = getMeth(type, paramType);

foreach (var m in methodsWithDefineParType)

{

info += "--> " + m.Name + "\n";

}

try

{

using (StreamWriter sw = new StreamWriter(path, false))

{

sw.WriteLine(info);

}

Console.WriteLine("Запись выполнена");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

static public object Create(string type, object[] values)

{

typeExist(type);

Type t = Type.GetType(type);

object result = Activator.CreateInstance(t, values);

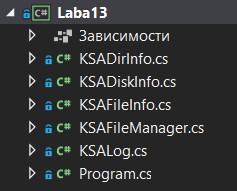
return result;

}

}

}

**№ 13 Работа с потоковыми классами и файловой системой**

****

**KSADirInfo.cs**

using System;

using System.IO;

// Так как нужно получить информацию об определенном директории, используем класс DirectoryInfo

namespace Laba13

{

class KSADirInfo

{

public delegate void InfoAboutAction(string action, DateTime date);

public static event InfoAboutAction Notify;

public static void dirInfoString(string path)

{

if (Directory.Exists(path))

{

int amountOfFiles = Directory.GetFiles(path).Length;

DateTime dateOfCreation = Directory.GetCreationTime(path);

int amoutOfSubdirs = Directory.GetDirectories(path).Length;

DirectoryInfo parent = Directory.GetParent(path);

string info = "========================== Путь к директорию: " + path + " ==========================\n" +

"Количество файлов в директории: " + amountOfFiles + "\n" +

"Дата создания: " + dateOfCreation + "\n" +

"Количество поддиректорий: " + amoutOfSubdirs + "\n" +

"Родительский директорий: " + parent + "\n";

Console.WriteLine(info);

}

else

{

Console.WriteLine("Указанного директория не существует\n");

}

KSADirInfo.Notify += KSALog.ActionsOfUser; // добавляем событию Notify обработчик

Notify("Вывод информации о директории " + path, DateTime.Now); // уведомляем обработчиков события о совершении действия

}

}

}

**KSADiskInfo.cs**

using System;

using System.IO;

namespace Laba13

{

class KSADiskInfo

{

public delegate void InfoAboutAction(string action, DateTime date);

public static event InfoAboutAction Notify;

public static void driveInfoString()

{

DriveInfo[] drives = DriveInfo.GetDrives();

string info = "";

foreach (DriveInfo dr in drives)

{

info += "========================== Диск: " + dr.Name + " ==========================\n" +

"Свободное место на диске (в байтах): " + dr.TotalFreeSpace + "\n" +

"Доступное место на диске (в байтах)" + dr.AvailableFreeSpace + "\n" +

"Общий размер диска (в байтах)" + dr.TotalSize + "\n" +

"Имя файловой системы: " + dr.DriveFormat + "\n" +

"Метка тома: " + dr.VolumeLabel + "\n";

}

Console.WriteLine(info);

KSADiskInfo.Notify += KSALog.ActionsOfUser; // добавляем событию Notify обработчик

Notify("Вывод информации о дисках", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

}

}

**KSAFileInfo.cs**

using System;

using System.IO;

namespace Laba13

{

class KSAFileInfo

{

public delegate void InfoAboutAction(string action, DateTime date);

public static event InfoAboutAction Notify;

public static void fileInfoString(string path)

{

FileInfo file = new FileInfo(path);

if (file.Exists)

{

string fullName = file.FullName;

long len = file.Length;

string ext = file.Extension;

string name = file.Name;

DateTime dateOfCreation = file.CreationTime;

DateTime dateOfLastChanging = file.LastWriteTime;

string info = "========================== Полное имя файла: " + fullName + " ==========================\n" +

"Имя файла: " + name + "\n" +

"Размер (в байтах): " + len + "\n" +

"Расширение: " + ext + "\n" +

"Дата создания: " + dateOfCreation + "\n" +

"Дата внесения последних изменений: " + dateOfLastChanging + "\n";

Console.WriteLine(info);

}

else

{

Console.WriteLine("Указанного файла не существует\n");

}

KSAFileInfo.Notify += KSALog.ActionsOfUser; // добавляем событию Notify обработчик

Notify("Вывод информации о файле " + path, DateTime.Now); // уведомляем обработчиков события о совершении действия

}

}

}

**KSAFileManager.cs**

using System;

using System.IO;

using System.Text;

using System.IO.Compression;

namespace Laba13

{

class KSAFileManager

{

public delegate void InfoAboutAction(string action, DateTime date);

public static event InfoAboutAction Notify;

public static void KSAInspect(string diskName)

{

KSAFileManager.Notify += KSALog.ActionsOfUser; // добавляем событию Notify обработчик

if (diskName.Length < 3)

{

Console.WriteLine("-> Введите имя диска в формате -> \"C:\"\", где C - имя диска");

return;

}

// проверяем существует ли диск с заданным именем

DriveInfo[] allDisks = DriveInfo.GetDrives();

bool exists = false;

foreach(DriveInfo d in allDisks)

{

if (d.Name == diskName)

exists = true;

}

// если не существует

if (!exists)

{

Console.WriteLine("-> Диска с именем " + diskName + " на данном компьютере не было найдено");

return;

}

// создаем объект, хранящий информацию о диске с именем diskName

DriveInfo disk = new DriveInfo(diskName);

if (disk.IsReady)

{

try

{

// получаем корневой директорий диска

DirectoryInfo rootDir = disk.RootDirectory;

string dirs = "Директории на диске " + diskName + ": ";

string files = "Файлы на диске " + diskName + ": ";

foreach (var dir in rootDir.GetDirectories())

{

dirs += "\n" + dir;

}

dirs += "\n";

foreach (var file in rootDir.GetFiles())

{

files += "\n" + file;

}

files += "\n";

string info = "========================== Информация о диске " + diskName + " ==========================\n" +

dirs + files;

string dirToCreate = @"D:\ООП\_2к\_1с\Laba13\KSAInspect";

DirectoryInfo newDir;

// проверяем, не существует ли уже директория с заданным именем

if (Directory.Exists(dirToCreate))

{

Console.WriteLine("-> Указанный директорий уже существует, новый файл будет создан в нем.");

}

else //если не существует, создаем его

{

newDir = Directory.CreateDirectory(dirToCreate);

Console.WriteLine("-> Директорий " + newDir.FullName + " создан");

Notify("-> Директорий " + newDir.FullName + " создан", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

// путь к файлу

string pathOfFile = dirToCreate + @"\KSADirInfo.txt";

using (FileStream fstream = new FileStream(pathOfFile, FileMode.OpenOrCreate))

{

// преобразуем строку в байты, поскольку FileStream представляет доступ к файлам на уровне байтов

byte[] infoInBytes = Encoding.Default.GetBytes(info);

// записываем текст в файл, если в нем была другая информация, она перезашется

fstream.Write(infoInBytes);

Console.WriteLine("-> Информация записана в файл");

Notify("Информация о диске " + diskName + " записана в файл " + pathOfFile, DateTime.Now); // уведомляем обработчиков события о совершении действия

}

string copyFilePath = @"D:\ООП\_2к\_1с\Laba13\driveInfo.txt";

if (!File.Exists(copyFilePath))

{

// true => целевой файл можно перезаписать

File.Copy(pathOfFile, copyFilePath, true);

Console.WriteLine("-> Файл " + pathOfFile + @" скопирован в директорий D:\ООП\_2к\_1с\Laba13 под именем driveInfo.txt");

Notify("Файл " + pathOfFile + @" скопирован в директорий D:\ООП\_2к\_1с\Laba13 под именем driveInfo.txt", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

else

{

File.Copy(pathOfFile, copyFilePath, true);

Console.WriteLine(@"-> Файл " + copyFilePath + " был перезаписан");

Notify(@"Файл " + copyFilePath + " был перезаписан", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

if (File.Exists(copyFilePath))

{

File.Delete(pathOfFile);

Console.WriteLine("-> Файл " + pathOfFile + " удален");

Notify("Файл " + pathOfFile + " удален", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

else

{

Console.WriteLine("-> Файл " + pathOfFile + " не был найден");

}

}

catch (Exception e)

{

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

}

}

}

public static void KSAExtensionCopy(string path)

{

try

{

if (!Directory.Exists(path))

{

Console.WriteLine("-> Указанного вами директория " + path + " не существует");

return;

}

string dirToCreate = @"D:\ООП\_2к\_1с\Laba13\KSAFiles";

DirectoryInfo newDir = new DirectoryInfo(dirToCreate);

// проверяем, не существует ли уже директория с заданным именем

if (Directory.Exists(dirToCreate))

{

Console.WriteLine("\n-> Указанный директорий уже существует, файлы будут добавлены в него");

}

else //если не существует, создаем его

{

newDir.Create();

Console.WriteLine("\n-> Директорий " + newDir.FullName + " был создан");

Notify("Директорий " + newDir.FullName + " был создан", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

// получаем полные пути файлов, соответствующих шаблону "\*.txt", где \* - любое сочетание символов

string[] files = Directory.GetFiles(path, "\*.txt");

foreach (string file in files)

{

// получаем только имя файла

string onlyFileName = Path.GetFileName(file);

File.Copy(file, Path.Combine(newDir.FullName, onlyFileName), true);

}

// Console.WriteLine(whereToMoveDir) -> D:\ООП\_2к\_1с\Laba13\KSAInspect\KSAFiles

// Console.WriteLine(newDir.FullName) -> D:\ООП\_2к\_1с\Laba13\KSAFiles

// чтобы переместить один директорий в другой последняя часть обоих путей должна совпадать, а части до существовать

// совпадающие части нужны для того же, для чего и указание имен файлов в обоих путях (исходном и итоговом) при их перемещении ->

// знать под каким именем должен быть получиться итоговый файл, а в нашем случае поддиректорий

// путь к директорию, куда нужно переместить директорию с файлами заданного расширения, где newDir.Name - имя конечного директория

string whereToMoveDir = Path.Combine(@"D:\ООП\_2к\_1с\Laba13\KSAInspect", newDir.Name);

if (Directory.Exists(whereToMoveDir))

{

Directory.Delete(whereToMoveDir, true);

Console.WriteLine("-> Директорий " + whereToMoveDir + " был удален");

Notify("Директорий " + whereToMoveDir + " был удален", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

newDir.MoveTo(whereToMoveDir);

Console.WriteLine("-> Директорий " + whereToMoveDir + " был создан");

Notify("Директорий " + whereToMoveDir + " был создан", DateTime.Now); // уведомляем обработчиков события о совершении действия

Console.WriteLine("-> Директорий " + newDir.FullName + " был перемещен в директорий " + whereToMoveDir + "\n");

Notify("Директорий " + newDir.FullName + " был перемещен в директорий " + whereToMoveDir, DateTime.Now); // уведомляем обработчиков события о совершении действия

}

catch (Exception e)

{

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

}

}

public static void MakeArchive()

{

string sourceFolder = @"D:\ООП\_2к\_1с\Laba13\KSAInspect\KSAFiles"; // исходная папка

string zipFile = @"D:\ООП\_2к\_1с\Laba13\KSAInspect\KSAFiles.zip"; // сжатый файл

string targetFolder = @"D:\ООП\_2к\_1с\Laba13\KSAInspect\unzipKSAFiles"; // папка, куда распаковывается файл

try

{

if (!Directory.Exists(sourceFolder))

{

Directory.CreateDirectory(sourceFolder);

}

if (File.Exists(zipFile))

{

File.Delete(zipFile);

}

ZipFile.CreateFromDirectory(sourceFolder, zipFile);

Console.WriteLine($"-> Папка {sourceFolder} архивирована в файл {zipFile}");

Notify($"Папка {sourceFolder} архивирована в файл {zipFile}", DateTime.Now); // уведомляем обработчиков события о совершении действия

if (Directory.Exists(targetFolder))

{

Directory.Delete(targetFolder, true);

}

ZipFile.ExtractToDirectory(zipFile, targetFolder);

Console.WriteLine($"-> Файл {zipFile} распакован в папку {targetFolder}\n");

Notify($"Файл {zipFile} распакован в папку {targetFolder}", DateTime.Now); // уведомляем обработчиков события о совершении действия

}

catch (Exception e)

{

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

}

}

}

}

**KSALog.cs**

using System;

using System.IO;

using System.Text;

namespace Laba13

{

class KSALog

{

public static void ActionsOfUser(string action, DateTime date)

{

try

{

string path = @"D:\ООП\_2к\_1с\Laba13\KSALogFile.txt";

using (StreamWriter swFile = new StreamWriter(path, true))

{

swFile.Write(action + " ->Время: " + date + "\n\n");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public static void ActionsForTheLastHour()

{

try

{

string path = @"D:\ООП\_2к\_1с\Laba13\KSALogFile.txt";

DateTime lastHour = DateTime.Now;

lastHour.AddHours(-1);

if (File.Exists(path))

{

int amountOfNotesForLastHour = 0;

if (File.GetLastWriteTime(path) < lastHour)

{

string notes = "";

using (StreamReader srFile = new StreamReader(path))

{

string line = srFile.ReadLine();

while (line != null)

{

amountOfNotesForLastHour++;

line = srFile.ReadLine();

notes += line + "\n";

}

amountOfNotesForLastHour /= 2; // в конце каждой записи добавлялся перенос строки

Console.WriteLine("Количество записей за последний час: " + amountOfNotesForLastHour);

Console.WriteLine("Записи за последний час" + notes);

}

}

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

**Program.cs**

namespace Laba13

{

class Program

{

static void Main(string[] args)

{

KSADiskInfo.driveInfoString();

KSADirInfo.dirInfoString(@"D:\open\_server\OpenServer\domains\chainsaw");

KSAFileInfo.fileInfoString(@"D:\ООП\_2к\_1с\Laba13\devilman\_crybaby.txt");

KSAFileManager.KSAInspect(@"C:\");

KSAFileManager.KSAExtensionCopy(@"D:\ООП\_2к\_1с\Laba13");

KSAFileManager.MakeArchive();

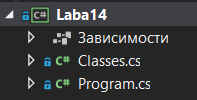
KSALog.ActionsForTheLastHour();

}

}

}

**№ 14 Сериализация**



**Classes.cs**

using System;

using System.Xml.Serialization;

namespace Laba14

{

[Serializable]

public class Company

{

public string company = "";

}

[XmlInclude(typeof(FeatureFilm))]

[Serializable]

public class Film

{

public string Title { get; set; }

public Company FilmCompany { get; set; }

public int DurationMinutes { get; set; }

[NonSerialized]

public float rating = 0;

}

[Serializable]

public class FeatureFilm : Film

{ }

}

**Program.cs**

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Text.Json;

using System.Threading.Tasks;

using System.Xml.Serialization;

using System.Xml;

using System.Xml.Linq;

using System.Linq;

namespace Laba14

{

class Program

{

static async Task Main(string[] args)

{

// 1

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

// --------------- BINARY ---------------

string path1 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\binary.txt";

Film f1 = new Film

{

Title = "Film 1",

DurationMinutes = 87,

rating = 3.7f,

FilmCompany = new Company() { company = "company 1" }

};

BinaryFormatter formatterBinary = new BinaryFormatter();

using (FileStream fs = new FileStream(path1, FileMode.OpenOrCreate))

{

formatterBinary.Serialize(fs, f1);

Console.WriteLine("-> Объект сериализован");

}

using (FileStream fs = new FileStream(path1, FileMode.OpenOrCreate))

{

Film desFilm = (Film)formatterBinary.Deserialize(fs);

Console.WriteLine("-> Объект десериализован");

Console.WriteLine($"Название: {desFilm.Title}\n" +

$"продолжительность: {desFilm.DurationMinutes}\n" +

$"рейтинг: {desFilm.rating}\n" +

$"компания производства: {desFilm.FilmCompany.company}\n");

}

// --------------- JSON ---------------

string path2 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\json.json";

FeatureFilm f2 = new FeatureFilm

{

Title = "Film 2",

DurationMinutes = 96,

rating = 3.3f,

FilmCompany = new Company() { company = "company 2" }

};

using (FileStream fs = new FileStream(path2, FileMode.OpenOrCreate))

{

// установка пробелов

var options = new JsonSerializerOptions

{

WriteIndented = true

};

await JsonSerializer.SerializeAsync<FeatureFilm>(fs, f2, options);

Console.WriteLine("-> Данные сохранены в файл");

}

using (FileStream fs = new FileStream(path2, FileMode.OpenOrCreate))

{

FeatureFilm objFromJson = await JsonSerializer.DeserializeAsync<FeatureFilm>(fs);

Console.WriteLine("-> Объект десериализован");

Console.WriteLine($"Название: {objFromJson.Title}\n" +

$"продолжительность: {objFromJson.DurationMinutes}\n" +

$"рейтинг: {objFromJson.rating}\n" +

$"компания производства: {objFromJson.FilmCompany.company}\n");

}

// --------------- XML ---------------

string path3 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\xml.xml";

Film f3 = new Film

{

Title = "Film 3",

DurationMinutes = 125,

rating = 4.9f,

FilmCompany = new Company() { company = "company 3" }

};

// прописываем поле rating, которое в классе Film нужно проигнорировать XmlIgnore = true (по умолчанию false)

XmlAttributeOverrides overrides = new XmlAttributeOverrides();

overrides.Add(typeof(Film), "rating", new XmlAttributes { XmlIgnore = true });

XmlSerializer formatterXml = new XmlSerializer(typeof(Film), overrides);

using (FileStream fs = new FileStream(path3, FileMode.OpenOrCreate))

{

formatterXml.Serialize(fs, f3);

Console.WriteLine("-> Объект сериализован");

}

using (FileStream fs = new FileStream(path3, FileMode.OpenOrCreate))

{

Film xmlFilm = (Film)formatterXml.Deserialize(fs);

Console.WriteLine("-> Объект десериализован");

Console.WriteLine($"Название: {xmlFilm.Title}\n" +

$"продолжительность: {xmlFilm.DurationMinutes}\n" +

$"рейтинг: {xmlFilm.rating}\n" +

$"компания производства: {xmlFilm.FilmCompany.company}\n");

}

// 2

Console.WriteLine("\n---------------------- СЕРИАЛИЗАЦИЯ МАССИВОВ ОБЪЕКТОВ ----------------------\n");

Film[] films = new Film[] { f1, f2, f3 };

// --------------- BINARY ---------------

string path4 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\arrayBin.txt";

BinaryFormatter binFormArray = new BinaryFormatter();

using (FileStream fs = new FileStream(path4, FileMode.OpenOrCreate))

{

binFormArray.Serialize(fs, films);

Console.WriteLine("-> Объекты сериализованы");

}

using (FileStream fs = new FileStream(path4, FileMode.OpenOrCreate))

{

Film[] desFilms = (Film[])binFormArray.Deserialize(fs);

Console.WriteLine("-> Объекты десериализованы");

foreach(var film in desFilms)

{

Console.WriteLine($"Название: {film.Title}\n" +

$"продолжительность: {film.DurationMinutes}\n" +

$"рейтинг: {film.rating}\n" +

$"компания производства: {film.FilmCompany.company}\n");

}

}

// --------------- JSON ---------------

string path5 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\arrayJson.json";

using (FileStream fs = new FileStream(path5, FileMode.OpenOrCreate))

{

// установка пробелов

var options = new JsonSerializerOptions

{

WriteIndented = true

};

await JsonSerializer.SerializeAsync<Film[]>(fs, films, options);

Console.WriteLine("-> Объекты сериализованы");

}

using (FileStream fs = new FileStream(path5, FileMode.OpenOrCreate))

{

Film[] objFromJson = await JsonSerializer.DeserializeAsync<Film[]>(fs);

Console.WriteLine("-> Объекты десериализованы");

foreach (var film in objFromJson)

{

Console.WriteLine($"Название: {film.Title}\n" +

$"продолжительность: {film.DurationMinutes}\n" +

$"рейтинг: {film.rating}\n" +

$"компания производства: {film.FilmCompany.company}\n");

}

}

// --------------- XML ---------------

string path6 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\arrayXml.xml";

XmlSerializer formatterArrayXml = new XmlSerializer(typeof(Film[]), overrides);

using (FileStream fs = new FileStream(path6, FileMode.OpenOrCreate))

{

formatterArrayXml.Serialize(fs, films);

Console.WriteLine("-> Объекты сериализованы");

}

using (FileStream fs = new FileStream(path6, FileMode.OpenOrCreate))

{

Film[] xmlFilms = (Film[])formatterArrayXml.Deserialize(fs);

Console.WriteLine("-> Объекты десериализованы");

foreach (var film in xmlFilms)

{

Console.WriteLine($"Название: {film.Title}\n" +

$"продолжительность: {film.DurationMinutes}\n" +

$"рейтинг: {film.rating}\n" +

$"компания производства: {film.FilmCompany.company}\n");

}

}

// 3

// XPATH

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

// XmlDocument представляет весь xml-документ

XmlDocument xDoc = new XmlDocument();

if (File.Exists(path6))

{

Console.WriteLine("Селектор 1 (//FilmCompany/company):");

xDoc.Load(path6);

var childNodes = xDoc.SelectNodes("//FilmCompany/company");

foreach(XmlNode node in childNodes)

{

Console.WriteLine(node.InnerText);

}

}

if (File.Exists(path6))

{

Console.WriteLine("\nСелектор 2 (FilmCompany[company = 'company 2']):");

xDoc.Load(path6);

var childNodes = xDoc.SelectNodes("//FilmCompany[company = 'company 2']");

foreach (XmlNode node in childNodes)

{

Console.WriteLine(node.InnerText);

}

}

// 4

// LINQ TO XML

Console.WriteLine("\n---------------------- LINQ TO XML ----------------------\n");

// Создание XML-документа

XDocument newDoc = new XDocument(

new XElement("students",

new XElement("student",

new XAttribute("ed\_form", "бюджет"),

new XElement("surname", "Хмельная"),

new XElement("first\_name", "Ольга"),

new XElement("patronymic", "Андреевна"),

new XElement("year\_of\_birth", 2001)

),

new XElement("student",

new XAttribute("ed\_form", "бюджет"),

new XElement("surname", "Зернович"),

new XElement("first\_name", "Ева"),

new XElement("patronymic", "Евгеньевна"),

new XElement("year\_of\_birth", 2004)

),

new XElement("student",

new XAttribute("ed\_form", "платная"),

new XElement("surname", "Лаканов"),

new XElement("first\_name", "Иван"),

new XElement("patronymic", "Георгиевич"),

new XElement("year\_of\_birth", 2002)

)

)

);

string path7 = @"D:\ООП\_2к\_1с\OOP\_Course2\_Term1\Laba14\xmlToLinq.xml";

newDoc.Save(path7);

Console.WriteLine("Xml-документ был создан\n");

// linq запросы

if (File.Exists(path7))

{

XDocument xmlLinqDoc = XDocument.Load(path7);

// 1 запрос (выбор студентов родившихся в 2002 и позже)

var birthSelection = from stud in xmlLinqDoc.Element("students").Elements("student")

where Convert.ToInt32(stud.Element("year\_of\_birth").Value) <= 2002

select stud;

foreach(var st in birthSelection)

{

Console.WriteLine(st.Element("surname").Value);

}

Console.WriteLine();

// 2 запрос (перебор определенных элементов и вывод их значений)

foreach(XElement st in xmlLinqDoc.Element("students").Elements("student"))

{

XAttribute nameAttr = st.Attribute("ed\_form");

XElement surnameEl = st.Element("surname");

XElement FirstNameEl = st.Element("first\_name");

Console.WriteLine($"Фамилия: {surnameEl.Value}\n" +

$"Имя: {FirstNameEl.Value}\n" +

$"Форма обучения: {nameAttr.Value}\n");

}

// 3 запрос (выбор студентов на бюджетной форме обучения)

var edFormSelection = from stud in xmlLinqDoc.Element("students").Elements("student")

where stud.Attribute("ed\_form").Value == "бюджет"

select stud;

foreach (var st in edFormSelection)

{

Console.WriteLine(st.Element("surname").Value);

}

Console.WriteLine();

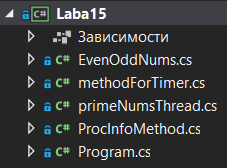
}

}

}

}

**№ 15 Работа с потоками выполнения**



**EvenOddNums.cs**

using System;

using System.IO;

using System.Threading;

namespace Laba15

{

partial class Program

{

//Используя средства синхронизации организуйте работу потоков, таким образом, чтобы

// a. выводились сначала четные, потом нечетные числа

// b. последовательно выводились одно четное, другое нечетное.

//// a

//// в качестве параметра объекта типа AutoResetEvent передаем false, что говорит о том,

//// что изначально объект будет в состоянии ожидания

//static AutoResetEvent waitHandler = new AutoResetEvent(false);

//public static void oddNums(int n)

//{

// waitHandler.WaitOne(); // поток ожидает передачи сигнального состояния объекту waitHandler

// string nums = "\nНечетные числа: \n";

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

// {

// if (i % 2 != 0)

// {

// Console.Write(i + "\t");

// nums += i + "\t";

// Thread.Sleep(10); // по условию скорость расчета чисел у потоков должна быть разная

// }

// }

// string path = @"D:\ООП\_2к\_1с\Laba15\evenOddNums.txt";

// using (StreamWriter swf = new StreamWriter(path, true))

// {

// swf.Write(nums);

// }

// waitHandler.Set();

//}

//public static void evenNums(int n)

//{

// string nums = "Четные числа: \n";

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

// {

// if (i % 2 == 0)

// {

// Console.Write(i + "\t");

// nums += i + "\t";

// Thread.Sleep(5); // по условию скорость расчета чисел у потоков должна быть разная

// }

// }

// string path = @"D:\ООП\_2к\_1с\Laba15\evenOddNums.txt";

// using (StreamWriter swf = new StreamWriter(path, true))

// {

// swf.Write(nums);

// }

// waitHandler.Set(); // передача сигнального состояния объекту waitHandler

//}

// b

// в качестве параметра объекта типа AutoResetEvent передаем false, что говорит о том,

// что изначально объект будет в состоянии ожидания

static AutoResetEvent waitHandler1 = new AutoResetEvent(true); // сразу в сигнальном состоянии

static AutoResetEvent waitHandler2 = new AutoResetEvent(false); // сазу в состоянии ожидания передачи сигнала

public static void oddNums(int n)

{

string path = @"D:\ООП\_2к\_1с\Laba15\alternation.txt";

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

{

if (i % 2 != 0)

{

// на протяженни 20 мс ожидаем получения потока waitHandler2, потом получаем его принудительно

// это нужно потому что вывод четных чисел завершится раньше нечетных и иначе программа просто остановиться,

// ожидая сигнальное состояние для waitHandler2, но его не получит, ибо метод вывода четных чисел,

// предоставляющий его, уже отработал

// вроде как, похоже на костыль, аха

waitHandler2.WaitOne(30);

Console.Write(i + "\t");

using (StreamWriter swf = new StreamWriter(path, true))

{

swf.Write(i + "\t");

}

Thread.Sleep(10);

waitHandler1.Set();

}

}

}

public static void evenNums(int n)

{

string path = @"D:\ООП\_2к\_1с\Laba15\alternation.txt";

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

{

if (i % 2 == 0)

{

waitHandler1.WaitOne(); // (\*) выполнение начинается здесь

Console.Write(i + "\t");

using (StreamWriter swf = new StreamWriter(path, true))

{

swf.Write(i + "\t");

}

Thread.Sleep(5);

waitHandler2.Set(); // освобождаем объект waitHandler2, который ожидает второй поток

}

}

}

}

}

**methodForTimer.cs**

using System;

namespace Laba15

{

partial class Program

{

public static void methodForTimer(object obj)

{

Console.Write("\tTimer!\t");

}

}

}

**primeNumsThread.cs**

using System;

using System.IO;

using System.Threading;

namespace Laba15

{

partial class Program

{

public static void primeNums(int n)

{

if (n < 1)

{

return;

}

string primeNumsStr = "";

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

{

bool isSimple = true;

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

{

if (i % j == 0)

{

isSimple = false;

break;

}

}

if (isSimple)

{

primeNumsStr += i + "\t";

Console.Write(i + "\t");

// приостанавливаем поток на 100 мс

Thread.Sleep(100);

};

}

string path = @"D:\ООП\_2к\_1с\Laba15\primeNums.txt";

using (StreamWriter swf = new StreamWriter(path))

{

swf.Write(primeNumsStr);

}

Console.WriteLine("\n\nЧисла были записаны в файл");

// получение и вывод информации о текущем потоке

Thread curThread = Thread.CurrentThread;

string primeNumsThreadInfo = $"\nИнформация о потоке:\n" +

$"Имя: {curThread.Name}\n" +

$"Идентификатор: {curThread.ManagedThreadId}\n" +

$"Выполняется ли поток: {curThread.IsAlive}\n" +

$"Приоритет: {curThread.Priority}\n";

Console.WriteLine(primeNumsThreadInfo);

}

}

}

**ProcInfoMethod.cs**

using System.Diagnostics;

using System.IO;

namespace Laba15

{

partial class Program

{

public static void procInfo()

{

Process[] processes = Process.GetProcesses();

string info = "Количество запущенных процессов: " + processes.Length + "\n";

int i = 1;

foreach (Process process in processes)

{

info += $"-> {i}\nИдентификатор процесса: {process.Id}\n" +

$"Имя процесса: {process.ProcessName}\n" +

$"Базовый приоритет: {process.BasePriority}\n";

i++;

}

string path = @"D:\ООП\_2к\_1с\Laba15\processesInfo.txt";

using (StreamWriter swf = new StreamWriter(path))

{

swf.Write(info);

}

System.Console.WriteLine("Данные о процессах записаны в файл\n");

}

}

}

**Program.cs**

using System;

using System.Reflection;

using System.Threading;

namespace Laba15

{

partial class Program

{

static void Main(string[] args)

{

// 1 (запись информации о запущенных процессах в файл)

procInfo();

// 2 (исследование текущего домена)

AppDomain curDom = AppDomain.CurrentDomain;

string curDomInfo = $"Имя домена приложения: {curDom.FriendlyName}\n" +

$"Каталог домена: {curDom.BaseDirectory}\n" +

$"Сборки домена:\n";

Assembly[] assemblies = curDom.GetAssemblies();

foreach (Assembly asm in assemblies)

{

curDomInfo += asm.GetName().Name + "\n";

}

Console.WriteLine(curDomInfo);

// 3

Console.Write("Введите число n для поиска простых чисел в диапазоне от 1 до n: ");

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

Thread primeNumsThread = new Thread(() => primeNums(n));

// запускаем в потоке primeNumsThread метод primeNums

primeNumsThread.Start();

// 4

Console.Write("Введите число n для поиска нечетных и четных чисел в диапазоне от 1 до n: ");

int odd\_even\_n = Convert.ToInt32(Console.ReadLine());

Thread oddThread = new Thread(() => oddNums(odd\_even\_n));

Thread evenThread = new Thread(() => evenNums(odd\_even\_n));

evenThread.Start();

oddThread.Start();

// 5

TimerCallback tm = new TimerCallback(methodForTimer);

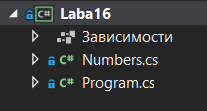
Timer timer = new Timer(tm, null, 0, 10);

}

}

}

**№ 16 Платформа параллельных вычислений**

****

**Numbers.cs**

using System;

using System.Threading;

using System.Threading.Tasks;

namespace Laba16

{

class Numbers

{

public static void matrixMult()

{

int rows1 = 0;

int cols1 = 0;

int rows2 = 0;

int cols2 = 0;

bool result;

Console.Write("Введите количество строк 1 матрицы: ");

result = int.TryParse(Console.ReadLine(), out rows1);

if (!result) {

Console.WriteLine("\nВведите число!");

return;

}

Console.Write("\nВведите количество столбцов 1 матрицы: ");

result = int.TryParse(Console.ReadLine(), out cols1);

if (!result)

{

Console.WriteLine("\nВведите число!");

return;

}

Console.Write("\nВведите количество столбцов 2 матрицы: ");

result = int.TryParse(Console.ReadLine(), out cols2);

if (!result)

{

Console.WriteLine("\nВведите число!");

return;

}

rows2 = cols1;

Random rnd = new Random();

Console.WriteLine("\n---------------------- 1 матрица ----------------------");

int[,] matr1 = new int[rows1, cols1];

for(int r1 = 0; r1 < rows1; r1++)

{

for(int c1 = 0; c1 < cols1; c1++)

{

matr1[r1, c1] = rnd.Next(-20, 100);

Console.Write(matr1[r1, c1] + "\t");

}

Console.WriteLine();

}

Console.WriteLine("\n---------------------- 2 матрица ----------------------");

int[,] matr2 = new int[rows2, cols2];

for (int r2 = 0; r2 < rows2; r2++)

{

for (int c2 = 0; c2 < cols2; c2++)

{

matr2[r2, c2] = rnd.Next(-20, 100);

Console.Write(matr2[r2, c2] + "\t");

}

Console.WriteLine();

}

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

int[,] multMatr = new int[rows1, cols2];

for (var i = 0; i < rows1; i++)

{

for (var j = 0; j < cols2; j++)

{

multMatr[i, j] = 0;

for (var k = 0; k < cols1; k++)

{

multMatr[i, j] += matr1[i, k] \* matr2[k, j];

}

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

}

Console.WriteLine();

}

}

public static void rowsCols(ref int rows1, ref int cols1, ref int rows2, ref int cols2)

{

bool result;

Console.Write("Введите количество строк 1 матрицы: ");

result = int.TryParse(Console.ReadLine(), out rows1);

if (!result)

{

Console.WriteLine("\nВведите число!");

return;

}

Console.Write("\nВведите количество столбцов 1 матрицы: ");

result = int.TryParse(Console.ReadLine(), out cols1);

if (!result)

{

Console.WriteLine("\nВведите число!");

return;

}

Console.Write("\nВведите количество столбцов 2 матрицы: ");

result = int.TryParse(Console.ReadLine(), out cols2);

if (!result)

{

Console.WriteLine("\nВведите число!");

return;

}

rows2 = cols1;

}

public static void matrixMultToken(CancellationToken token, int rows1, int cols1, int rows2, int cols2)

{

Random rnd = new Random();

Console.WriteLine("\n---------------------- 1 матрица ----------------------");

int[,] matr1 = new int[rows1, cols1];

for (int r1 = 0; r1 < rows1; r1++)

{

for (int c1 = 0; c1 < cols1; c1++)

{

matr1[r1, c1] = rnd.Next(-20, 100);

Console.Write(matr1[r1, c1] + "\t");

}

Console.WriteLine();

}

Console.WriteLine("\n---------------------- 2 матрица ----------------------");

int[,] matr2 = new int[rows2, cols2];

for (int r2 = 0; r2 < rows2; r2++)

{

for (int c2 = 0; c2 < cols2; c2++)

{

matr2[r2, c2] = rnd.Next(-20, 100);

Console.Write(matr2[r2, c2] + "\t");

}

Console.WriteLine();

}

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

int[,] multMatr = new int[rows1, cols2];

for (var i = 0; i < rows1; i++)

{

if (token.IsCancellationRequested)

{

Console.WriteLine("Операция прервана");

return;

}

for (var j = 0; j < cols2; j++)

{

multMatr[i, j] = 0;

for (var k = 0; k < cols1; k++)

{

multMatr[i, j] += matr1[i, k] \* matr2[k, j];

}

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

Thread.Sleep(2000);

}

Console.WriteLine();

}

}

public static int Task1()

{

return 1;

}

public static int Task2()

{

return 2;

}

public static int Task3()

{

return 3;

}

public static void genArray(int amount)

{

Random rnd = new Random();

int[] array = new int[amount];

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

{

array[i] = rnd.Next(-100, 100);

}

foreach (int n in array)

{

Console.Write(n + "\t");

}

}

public static async void asyncMethod()

{

await Task.Run(() =>

{

Thread.Sleep(4000);

Console.WriteLine("Асинхронный метод завершился!");

});

}

}

}

**Program.cs**

using System;

using System.Threading.Tasks;

using System.Threading;

using System.Diagnostics;

using System.Collections.Generic;

using System.Collections.Concurrent;

namespace Laba16

{

class Program

{

static void Main(string[] args)

{

// 1 создайте длительную по времени задачу (на основе Task) на выбор: перемножение матриц

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Task matrMultTask = new Task(Numbers.matrixMult);

// Оцените производительность выполнения используя объект Stopwatch на нескольких прогонах.

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

matrMultTask.Start();

matrMultTask.Wait();

stopWatch.Stop();

TimeSpan ts = stopWatch.Elapsed;

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("\n-> RunTime " + elapsedTime);

// Выведите идентификатор текущей задачи, проверьте во время выполнения – завершена ли задача и выведите ее статус.

Console.WriteLine($"\nИнформация о задаче: \n" +

$"Идентификатор: {matrMultTask.Id}\n" +

$"Статус: {matrMultTask.Status}\n");

// 2 Реализуйте второй вариант этой же задачи с токеном отмены CancellationToken и отмените задачу.

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

CancellationTokenSource cancelTokenSource = new CancellationTokenSource();

CancellationToken token = cancelTokenSource.Token;

// так как нам необходимо, чтобы пользователь ввел некоторые значения, а в этот ввод,

// если оставить его в функции matrixMultTokenб может вмешаться основной поток, создаем метод

// с помощью которого плучаем и проверяем значения на корректность

int rows1 = 0;

int cols1 = 0;

int rows2 = 0;

int cols2 = 0;

Task.Run(() => Numbers.rowsCols(ref rows1, ref cols1, ref rows2, ref cols2)).Wait();

Task matrMultTaskWithToken = new Task(() => Numbers.matrixMultToken(token, rows1, cols1, rows2, cols2));

matrMultTaskWithToken.Start();

Console.WriteLine("\n-> Введите Y для отмены операции <-\n");

string s = Console.ReadLine();

if (s == "Y" || s == "y") { }

cancelTokenSource.Cancel();

// 3 Создайте три задачи с возвратом результата и используйте их для выполнения четвертой задачи. Например, расчет по формуле.

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Task<int>[] tasks = new Task<int>[3]

{

new Task<int>(Numbers.Task1),

new Task<int>(Numbers.Task2),

new Task<int>(Numbers.Task3)

};

foreach (var t in tasks)

{

t.Start();

}

int sum = 0;

foreach (var t in tasks)

{

sum += t.Result;

}

Console.WriteLine("Сумма 1 + 2 + 3 = " + sum);

// 4 Создайте задачу продолжения (continuation task) в двух вариантах

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

// 1) C ContinueWith -планировка на основе завершения множества предшествующих задач

Task<int> task4\_1 = new Task<int>(() => 1);

Task display = task4\_1.ContinueWith(num => Console.WriteLine(

$"-> Задача с идентификатором \"{task4\_1.Id}\" вернула значение {num.Result}"));

task4\_1.Start();

// 2) На основе объекта ожидания и методов GetAwaiter(),GetResult();

Task<int> task4\_2 = Task.Run(() => 2);

// ждем завершения task4\_2

var awaiter = task4\_2.GetAwaiter();

// получив объект ожидания, решаем, что будем делать дальше; в данном случае выведем значение, которое вернула задача task4\_2

awaiter.OnCompleted(() => Console.WriteLine(

$"-> Задача с идентификатором \"{task4\_2.Id}\" вернула значение {awaiter.GetResult()}"));

Thread.Sleep(100);

// 5 Используя Класс Parallel распараллельте вычисления циклов For(), ForEach().

// Оцените производительность по сравнению с обычными циклами

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 5 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

int amount = 5;

Console.Write("Генерация 3 массивов по 5 элементов с помощью Parallel.For: ");

stopWatch.Start();

Parallel.For(1, 140, (num) => Numbers.genArray(amount));

stopWatch.Stop();

ts = stopWatch.Elapsed;

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("\n-> RunTime " + elapsedTime);

Console.Write("\nГенерация 3 массивов по 5 элементов с помощью цикла запуска задач: ");

stopWatch.Start();

for (int i = 1; i < 140; i++)

{

Task.Run(() => Numbers.genArray(amount)).Wait();

}

stopWatch.Stop();

ts = stopWatch.Elapsed;

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("\n-> RunTime " + elapsedTime);

List<int> list = new List<int>() { 3, 4, 7, 8, 9, 2, 4, 7, 12, 9, 5, 5, 4, 23, 80, 65 };

Console.Write("\nГенерация массивов по 3, 4, 7, 8, 9, 2, 4, 7, 12, 9, 5, 5, 4, 23, 80, 65 элементов с помощью Parallel.ForEach: ");

stopWatch.Start();

ParallelLoopResult result = Parallel.ForEach(list, Numbers.genArray);

stopWatch.Stop();

ts = stopWatch.Elapsed;

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("\n-> RunTime " + elapsedTime);

Console.Write("\nГенерация массивов по 3, 4, 7, 8, 9, 2, 4, 7, 12, 9, 5, 5, 4, 23, 80, 65 элементов с помощью цикла запуска задач: ");

stopWatch.Start();

for (int i = 0; i < list.Count; i++)

{

Task.Run(() => Numbers.genArray(list[i])).Wait();

}

stopWatch.Stop();

ts = stopWatch.Elapsed;

elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine("\n-> RunTime " + elapsedTime);

// 6 Используя Parallel.Invoke() распараллельте выполнение блока операторов.

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 6 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Parallel.Invoke(

() =>

{

Thread.Sleep(1000);

Console.WriteLine("Я поспал 1 секунду!");

},

() => Console.WriteLine("А куда спать?"),

() =>

{

for (int i = 1; i <= 3; i++)

{

Console.WriteLine("Цикл на итерации: " + i);

}

}

);

// 7 Используя Класс BlockingCollection реализуйте следующую задачу:

// Есть 5 поставщиков бытовой техники, они завозят уникальные товары на склад(каждый по одному) и

// 10 покупателей – покупают все подряд, если товара нет - уходят.В вашей задаче: cпрос превышает предложение.

// Изначально склад пустой.У каждого поставщика своя скорость завоза товара.Каждый раз при изменении состоянии склада

// выводите наименования товаров на складе.

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 7 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

BlockingCollection<string> bc = new BlockingCollection<string>(5);

int tea = 1;

// псотавщики

Task[] shippers = new Task[5]

{

new Task(() =>

{

while (!bc.IsAddingCompleted)

{

bc.Add("бензопила");

Console.WriteLine("На склад поставлена бензопила");

Thread.Sleep(1000);

}

}

),

new Task (()=>

{

// когда будет доставлено 3 чая, поток заснет на 1 секунду, после чего заблокируется возможность добавления товаров

while (tea <= 3 && !bc.IsAddingCompleted)

{

bc.Add("чай");

Console.WriteLine("На склад поставлен чай");

tea++;

Thread.Sleep(1000);

}

bc.CompleteAdding();

}

),

new Task (()=>

{

while (!bc.IsAddingCompleted)

{

bc.Add("странный кот");

Console.WriteLine("На склад поставлен странный кот");

Thread.Sleep(1000);

}

}

),

new Task (()=>

{

while (!bc.IsAddingCompleted)

{

bc.Add("Сатурн");

Console.WriteLine("На склад поставлен Сатурн");

Thread.Sleep(1000);

}

}

),

new Task (()=>

{

while (!bc.IsAddingCompleted)

{

bc.Add("семизубый меч");

Console.WriteLine("На склад поставлен семизубый меч");

Thread.Sleep(1000);

}

}

)

};

string good = "";

Task[] consumers = new Task[10]

{

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

new Task(()=>

{

while (!bc.IsCompleted)

{

if(bc.TryTake(out good))

Console.WriteLine("Взят товар с наименованием: " + good);

else

Console.WriteLine("Покупатель ушёл, ничего не купив");

Thread.Sleep(400);

}

}

),

};

foreach (var sр in shippers)

{

sр.Start();

}

foreach (var c in consumers)

{

c.Start();

}

Console.ReadLine();

// 8 Используя async и await организуйте асинхронное выполнение любого метода.

Console.WriteLine("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 8 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Numbers.asyncMethod();

Console.WriteLine("Поток Main не заблокирован асинхронным методом!");

Console.ReadLine();

}

}

}