Оглавление

[1.Типы данных, объявление переменной, var и dynamic 3](#_Toc124203354)

[2.Ввод/вывод 3](#_Toc124203355)

[3.Приведение типов 3](#_Toc124203356)

[4.Boxing/unboxing 3](#_Toc124203357)

[5.Строки 3](#_Toc124203358)

[6.Массивы 5](#_Toc124203359)

[7.Кортежи 6](#_Toc124203360)

[8.Функция с кортежами 6](#_Toc124203361)

[9.Классы, конструкторы, свойства 6](#_Toc124203362)

[10. Класс System.Object и переопределение его методов 7](#_Toc124203363)

[11. Перегрузка операторов 8](#_Toc124203364)

[11. Интерфейс, абстрактный класс, класс sealed 8](#_Toc124203365)

[12. Структуры, перечисления, класс partial 9](#_Toc124203366)

[13. Исключения 9](#_Toc124203367)

[14. Debug, debugger 9](#_Toc124203368)

[15. Обобщения 10](#_Toc124203369)

[16. Лямбда выражение 11](#_Toc124203370)

[17. foreach 11](#_Toc124203371)

[18. События и делегаты 11](#_Toc124203372)

[19. Коллекции, стандартные интерфейсы 12](#_Toc124203373)

[20. LINQ 13](#_Toc124203374)

[20. Рефлексия 14](#_Toc124203375)

[21. Файловая система 16](#_Toc124203376)

[22. Запись/чтение файлов 18](#_Toc124203377)

[23. Время/дата 18](#_Toc124203378)

[24. Секундомер 20](#_Toc124203379)

[25. Сериализация/десериализация 20](#_Toc124203380)

[26. XML 21](#_Toc124203381)

[27. Потоки 22](#_Toc124203382)

[28. Timer 25](#_Toc124203383)

[29. Task 25](#_Toc124203384)

[30. Структура main 27](#_Toc124203385)

[31. Random 27](#_Toc124203386)

[32. Паттерны проектирования. Паттерн Singleton. 28](#_Toc124203387)

[33. Паттерны проектирования. Паттерн Factory method. 29](#_Toc124203388)

[34. Паттерны проектирования. Паттерн Abstract Factory. 30](#_Toc124203389)

[35. Паттерны проектирования. Паттерн Prototype. 33](#_Toc124203390)

[36. Паттерны проектирования. Паттерн Builder. 34](#_Toc124203391)

[37. Паттерны проектирования. Паттерн Adapter. 34](#_Toc124203392)

[38. Паттерны проектирования. Паттерн Decorator. 36](#_Toc124203393)

[39. Паттерны проектирования. Паттерн State. 37](#_Toc124203394)

[40. Паттерны проектирования. Паттерн Command. 39](#_Toc124203395)

[41. Паттерны проектирования. Паттерн Memento. 40](#_Toc124203396)

[42. Паттерны проектирования. Паттерн Observer. 41](#_Toc124203397)

[43. Паттерны проектирования. Паттерн strategy. 42](#_Toc124203398)

# 1.Типы данных, объявление переменной, var и dynamic

bool check = true;

byte bMin = byte.MinValue;

int iMax = int.MaxValue; //4 byte

float fMin = float.MinValue; //4 byte

long lMin = long.MinValue; //8 byte

decimal deMin = decimal.MinValue; //5...17byte

object b = 184; //4 - 32, 8 – 64

var n1 = 66;

var n2 = 12.5;

var str\_v = "Hello!";

dynamic val1 = "GeeksforGeeks";

# 2.Ввод/вывод

Console.WriteLine("Строка: {0}", str);

Console.WriteLine($"{shMax}");//shMax-переменная

str = Console.ReadLine();

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

# 3.Приведение типов

string arrrfe = "1235";

int coe = Convert.ToInt32(arrrfe);

string arrrfe = "1235";

int coe = int.Parse(arrrfe);

# 4.Boxing/unboxing

//упаковка ---------- сохраняет его в управляемой куче (хипе)

int i = 123;

object o = i;

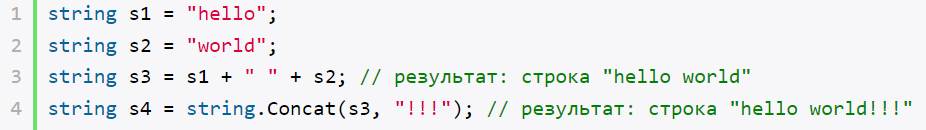
//распаковка

o = 123;

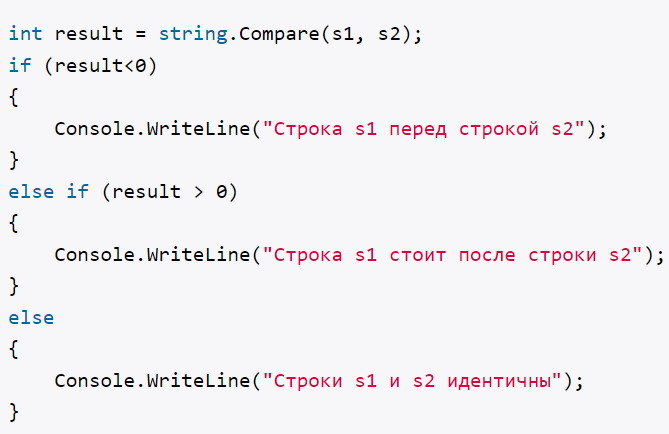
i = (int)o;

# 5.Строки

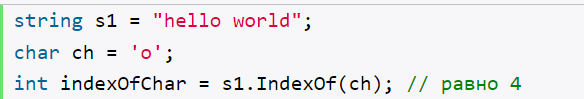
* Конкатенация строк:



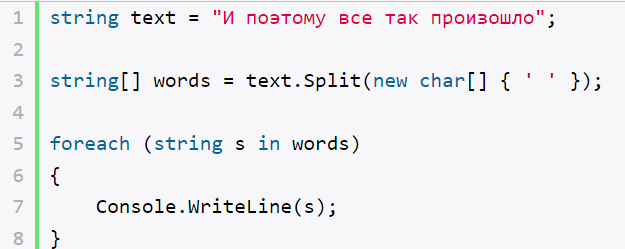
* Сравнение строк



* Поиск в строке



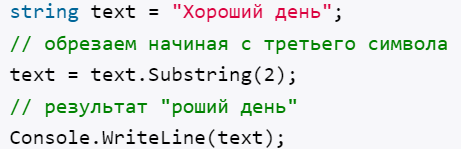
* Разделение строк



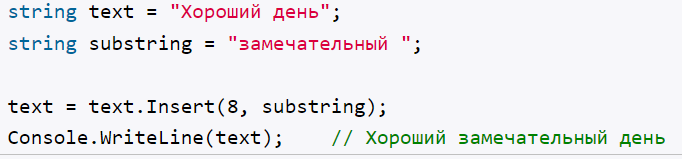
С удалением лишних пробелов



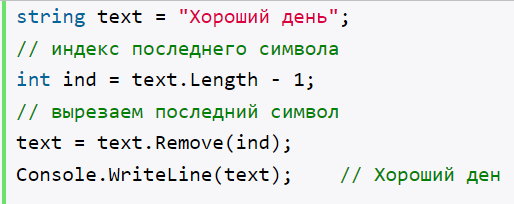
* Обрезка строки



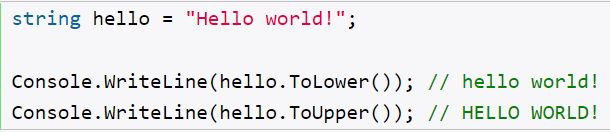
* Вставка



* Удаление строк



* Смена регистра



**Динамические строки: StringBuilder *(можно после создания изменять)***

StringBuilder sb = new StringBuilder("Привет мир"); //можно изменять как в С++ по индексу

# 6.Массивы

int[,] numbers = { { 1, 2, 3 }, { 4, 5, 6 } }; //матрица

/\*ТАК ЖЕ МОЖНО:

int[,] nums1;

int[,] nums2 = new int[2, 3];

int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };

\*/

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

{

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

{

Console.Write($"{numbers[iv, j]} \t");

}

Console.WriteLine();

}

string[] people = { "Tom", "Sam", "Bob" }; //одномерный массив

Console.WriteLine("\nДлина массива: {0}", people.Length);

double[][] myArr = new double[3][]; //ступенчатый массив

myArr[0] = new double[2];

myArr[1] = new double[3];

myArr[2] = new double[4];

# 7.Кортежи

(int, string, char, string, ulong) kortezh = (19, "Hi", 'F', "world", 9999999);

Console.WriteLine("\nКортеж целиком: {0}", kortezh);

Console.WriteLine("Элементы кортежа 1, 3, 4: {0} {1} {2}", kortezh.Item1, kortezh.Item3, kortezh.Item4);

var (first, second, third, four, five) = kortezh; //можно вынести var за скобки

# 8.Функция с кортежами

(int, int, int, char) Localfunction(int[] numbers, string str1)

{

int max = 0;

int min = Int32.MaxValue;

int sum = 0;

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

{

if (numbers[i] > max)

{

max = numbers[i];

}

if (numbers[i] < min)

{

min = numbers[i];

}

sum += numbers[i];

}

char smb = str1[0];

var tuple1 = (max, min, sum, smb);

return tuple1;

}

# 9.Классы, конструкторы, свойства

class PHONE{

…

}

public PHONE() //конструктор без параметров

{

ID = Guid.NewGuid().ToString();

name\_1 = "-";

surname\_1 = "-";

credit\_1 = "0";

time\_1 = "0" + " минут(ы)";

time\_2 = "0" + " минут(ы)";

count++;

}

public PHONE(string a, string n, string sn) //конструктор с параметрами

{

ID = Guid.NewGuid().ToString();

name\_1 = n;

surname\_1 = sn;

count++;

}

private PHONE() { } //закрытый конструктор

static PHONE()//статический конструктор

{

mark\_of\_phone = "iphone";

}

private string credit;

public string credit\_1 //свойство

{

set

{

try

{

int f = int.Parse(value);

if (f < 0)

{

credit = null;

}

else

{

credit = value;

}

}

catch

{

credit = null;

}

}

get

{

return credit;

}

}

# 10. Класс System.Object и переопределение его методов

public override string? ToString()

{

return "Это переопределенный метод ToString";

}

public string Name { get; set; } = "";

public override int GetHashCode()

{

return Name.GetHashCode();

}

public string Name { get; set; } = "";

public override bool Equals(object? obj)

{

// если параметр метода представляет тип Person

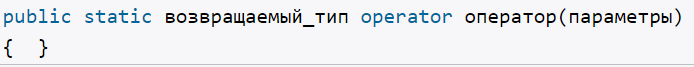
// то возвращаем true, если имена совпадают

if (obj is Person person) return Name == person.Name;

return false;

}

# 11. Перегрузка операторов



class Counter

{

public int Value { get; set; }

public static Counter operator +(Counter counter1, Counter counter2)

{

return new Counter { Value = counter1.Value + counter2.Value };

}

public static bool operator >(Counter counter1, Counter counter2)

{

return counter1.Value > counter2.Value;

}

public static bool operator <(Counter counter1, Counter counter2)

{

return counter1.Value < counter2.Value;

}

}

public static bool operator <(Set<T> set1, Set<T> set2) => Subset(set1, set2);

/// Подмножество ПЕРЕГРУЗКА.

public static bool Subset(Set<T> set1, Set<T> set2)

{

if (set1 == null)

{

throw new ArgumentNullException(nameof(set1));

}

if (set2 == null)

{

throw new ArgumentNullException(nameof(set2));

}

/\* Перебираем элементы первого множества.

Если все элементы первого множества -> во втором -

это подмножество. Возвращаем истину, иначе ложь. \*/

var result = set1.\_items.All(s => set2.\_items.Contains(s));

return result;

}

# 11. Интерфейс, абстрактный класс, класс sealed

interface INFORMATION

{

string age(string age);

void name(string name);

}

public abstract class TV\_program

{

public abstract void start\_time(string time);

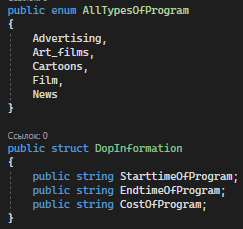
public abstract void duration(string dur);

}



Sealed – запрещает наследование

# 12. Структуры, перечисления, класс partial

Partial – использование одного класса в нескольких файлах(например)

# 13. Исключения

class WeekException : Exception

{

public string value;

public WeekException(string message, string val) : base(message)

{

value = val;

}

}

ArtFilm artfilm\_2 = new("В мире БГТУ", "Новиков Роман", 2022);

try

{

artfilm\_2.ProgramGuide("Вторник");

}

catch (WeekException ex)

{

Console.WriteLine("Ошибка: {0}", ex.Message);

Console.WriteLine($"Некорректное значение : {ex.value}");

}

finally

{

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

}

# 14. Debug, debugger

Debug.Write("Data"); //send message to debugger

Debug.WriteLine(23 \* 34);

int x = 5, y = 3;

Debug.WriteIf(x > y, "x is greater than y");

Debugger.Launch(); //choose of debugger

Debugger.Break(); //==breakpoint

# 15. Обобщения

interface IItems <T>

{

// вывод массива

void Print(string message);

// Добавить элемент в конец массива

void Add(T value);

// Удалить элемент с позиции index

void Delete(int index);

}

class ArrayItems<T> : IItems<T>

{

T[] items;

public ArrayItems()

{

// Установить значение по умолчанию

items = default(T[]);

}

public ArrayItems(T[] \_items)

{

items = \_items;

}

// Добавить элемент в конец массива

public void Add(T value)

{

T[] items2 = items;

items = new T[items.Length + 1];

for (int i = 0; i < items.Length - 1; i++)

items[i] = items2[i];

items[items.Length - 1] = value;

}

…

public class Set <T> : IEnumerable<T>, IItems<T> //перечислитель

{

private List<T> \_items = new();

public int Count => \_items.Count; //размерность

public void Add(T item)

{

if (item == null)

{

throw new NULLException("вы ничего не передали!", item);

}

// множество содержит только уникальные элементы

if (!\_items.Contains(item))

{

\_items.Add(item);

}

}

var set1 = new Set<int>()

{

-1, 2, 3, 4, 5, -6

};

# 16. Лямбда выражение

Message hello = () =>

{

Console.Write("Hello ");

Console.WriteLine("World");

};

hello(); // Hello World

# 17. foreach

foreach (var item in set2.\_items)

{

if (set1.\_items.Contains(item))

{

resultSet.Add(item);

}

}

# 18. События и делегаты

public delegate void deleg1(int count);

public delegate void deleg2();

public delegate void deleg3();

public static event deleg1 Upgrade;

public static event deleg2 TurnOn;

public static event deleg3 Throw\_out;

…

Boss microwave = new("Микроволновка", "150", 15);

Boss PC = new("Компьютер", "215", 5);

Boss hoover = new("Пылесос", "100", 10);

Boss TV = new("Телевизор", "290", 22);

Upgrade += microwave.upgrade;

Upgrade += PC.upgrade;

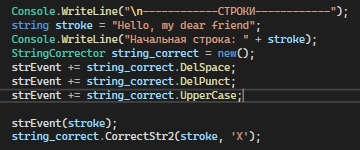
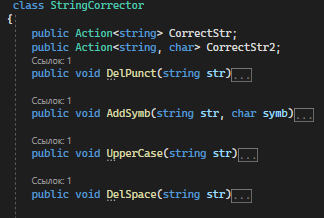
TurnOn += hoover.turnOn;

Throw\_out += TV.ThrowOut;

Upgrade(25);

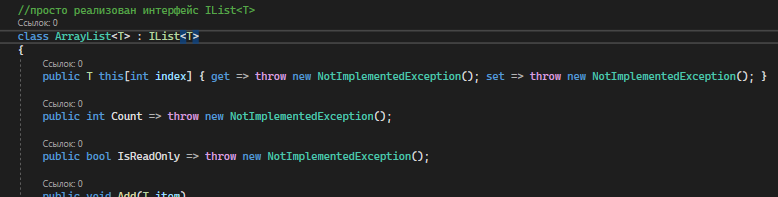
TurnOn();

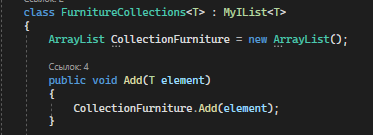
Throw\_out();

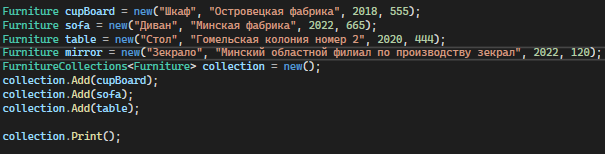


# 19. Коллекции, стандартные интерфейсы

\*Создание своей коллекции на базе существующей







Варианты стандартных коллекций (через [название коллекции]. можно изучить методы каждой):

**Dictionary<Tkey, TValue> *(ключ-значение)***, **HashSet<T>*(хеш-таблица),* LinkedList<T>*(двунаправленный список)***, **List<T>*(динамический массив)***, **Queue<T>, Stack<T> …**

**Стандартные интерфейсы:**

**ICollection<T>**

Определяет основополагающие свойства обобщенных коллекций

**IComparer<T>**

Определяет обобщенный метод Compare() для сравнения объектов, хранящихся в коллекции

**IEnumerable<T>**

Определяет обобщенный метод GetEnumerator(), предоставляющий перечислитель для любого класса коллекции

interface IComparable

{

int CompareTo( object obj )

}

Метод должен возвращать:

- 0, если текущий объект и параметр равны;

- отрицательное число, если текущий объект меньше параметра;

- положительное число, если текущий объект больше параметра.

# 20. LINQ

var findMonth = from m in month

where m.Length == len

select m;

Console.WriteLine("\nМесяца с заданным количеством символов (" + len + "):");

foreach (var m in findMonth)

{

Console.Write(m);

}

Console.WriteLine("\n\nЛетние и зимние месяца: ");

var SummerAndWinter = from sw in month

where sw is ( "January" or

"February" or

"December" or

"June" or

"July" or

"August" )

select sw;

foreach (var s in SummerAndWinter)

{

Console.Write(s + ' ');

}

Console.WriteLine("\n\nМесяца в алфaвитном порядке:");

var sortmonth = from p in month orderby p select p;

var searchDebet = (from d in abonents where d.debet == deb select d).Count();

Console.WriteLine("Количество абонентов cо значением дебета " + deb + ": " + searchDebet +"\n");

var MaxOfCityCalls = (from mc in abonents select mc.TimeOfcityCalls).Max();

//агрегирование

var averageCalls = (from ac in abonents select ac.TimeOfcityCalls).Average();

//группировка

var noIC = abonents.GroupBy(ic => ic.TimeOfInternationalCalls > 0);

Console.WriteLine("Группировка: ");

foreach (var mc in noIC)

{

if(mc.Key == true)

{

Console.WriteLine("Абоненты, которые использовали междугородную связь: ");

}

else

Console.WriteLine("Абоненты, которые НЕ использовали междугородную связь: ");

foreach(var mc2 in mc)

{

Console.WriteLine(mc2.surname);

}

Console.WriteLine();

}

//кванторы

string sm = "Kravchenko";

Console.WriteLine("Поиск указанной фамилии: ");

var searchSurname = abonents.Any(ss => ss.surname == sm);

if (searchSurname == false)

Console.WriteLine("Нет абонента с фамиилей "+sm+'\n');

else

Console.WriteLine("Абонент с фамилией " + sm + " присутствует!\n");

//разбиение

var result = abonents.Take(3);

Console.WriteLine("Первых ТРИ абонента:");

foreach(var r in result)

{

r.ToString();

}

//join

abonentInfo[] abon =

{

new abonentInfo("Aleksey", "Belarus"), new abonentInfo("Thomas", "Germany"),

new abonentInfo("Polina", "Belarus"), new abonentInfo("Mike", "England")

};

TelecommunicationOperator[] TO =

{

new TelecommunicationOperator("Belarus", "+375"), new TelecommunicationOperator("Germany", "+49"),

new TelecommunicationOperator("England", "+44")

};

var connection = from a in abon

join t in TO on a.country equals t.country

select new { name = a.name, country = t.country, codeOfTheCountry = t.CodeOFTheCountry };

# 20. Рефлексия

using System.Reflection;

static class Reflector<T> where T : class, new()

{

private static Type type = typeof(T);

Console.WriteLine($"Информация про сборку: {type.Assembly.FullName}.");

if(type.IsPublic)

{

Console.WriteLine("\nПубличный конструктор присутствует!\n");

}

Console.WriteLine("\nПубличные методы:");

int count = 0;

MethodInfo[] methods = type.GetMethods (BindingFlags.Public | BindingFlags.Instance | BindingFlags.DeclaredOnly);

foreach (MethodInfo method in methods)

{

Console.WriteLine($"Название метода: {method.Name}");

}

Console.WriteLine("\nИнформация про поля и свойства:");

FieldInfo[] fields = type.GetFields(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Static);

foreach (var field in fields)

{

i++;

nameType = field.FieldType.ToString(); // получить название типа поля field структуры Worker

Console.Write($"\nПоле(свойство)[{i}] = {field.Name}, тип = {nameType}, модификатор доступа: ");

if (field.IsStatic)

{

Console.Write("static");

}

}

Type[] interfaces = type.GetInterfaces();

int i = 0;

foreach (Type t in interfaces)

{

Console.WriteLine("\nИнтерфейс[{0}] = {1}", i, t.Name);

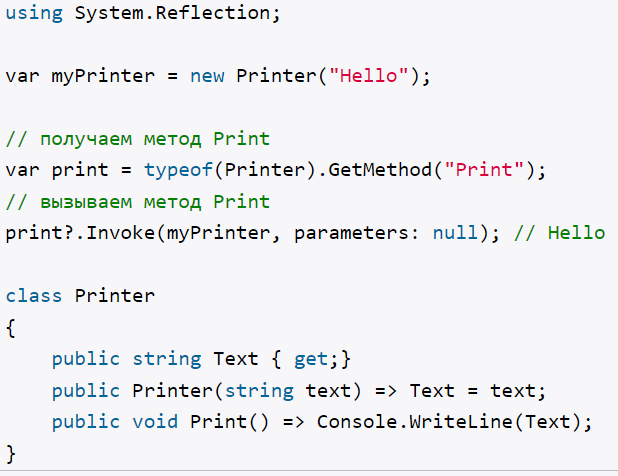
}

var foo = type.GetMethods().Where(i => i.GetParameters().Any(item => item.ParameterType == param));

ParameterInfo[] parameters = methodInfo.GetParameters();

INVOKE:





# 21. Файловая система

DriveInfo[] drives = DriveInfo.GetDrives();

foreach (DriveInfo drive in drives)

{

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

Console.WriteLine($"Тип: {drive.DriveType}");

if (drive.IsReady)

{

Console.WriteLine($"Объем диска: {drive.TotalSize}");

Console.WriteLine($"Свободное пространство: {drive.TotalFreeSpace}");

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

}

Console.WriteLine();

}

}

FileInfo f = new (nameFile);

Console.WriteLine("Путь к файлу: " + f.FullName);

FileInfo file = new(nameCatalog);

long size = file.Length;

Console.WriteLine("Размер файла: " + size);

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

Console.WriteLine("Название файла: " + file.Name);

DateTime fileCreatedDate = File.GetCreationTime(nameFile);

Console.WriteLine("Дата и время создания: " + fileCreatedDate);

DateTime ChangeFile = File.GetLastWriteTime(nameFile);

Console.WriteLine("Последнее изменение файла: " + ChangeFile);

DirectoryInfo di = new (NameDir);

FileInfo[] fiArr = di.GetFiles();

Console.WriteLine("Количество файлов: " + fiArr.Count());

DateTime creationTime = Directory.GetCreationTime(nameDir);

Console.WriteLine("Время и дата создания директория: " + creationTime);

DirectoryInfo[] diArr = di.GetDirectories();

Console.WriteLine("Количество поддиректориев: " + diArr.Count());

Console.WriteLine("Родительские директории: ");

Console.WriteLine(Directory.GetParent(pathToFile));

Console.WriteLine($"Файлы диска {NameDisk} :");

foreach (FileInfo fi in fiArr)

Console.WriteLine(fi);

Console.WriteLine($"\nПапки диска {NameDisk} :");

DirectoryInfo[] fiArr2 = di.GetDirectories();

foreach (DirectoryInfo fi in fiArr2)

Console.WriteLine(fi);

DirectoryInfo dirInfo = new(NameDir);

if (!dirInfo.Exists)

{

dirInfo.Create();

}

using (StreamWriter writer = new(NameFile, false))

{

writer.WriteLine("Сохранённая информация)");

}

FileInfo fi = new(pathTo);

if (!fi.Exists)

File.Copy(pathFrom, pathTo);

DirectoryInfo dirFrom = new(pathFrom);

DirectoryInfo dirTo = new(nameDir);

var files = dirFrom.GetFiles();

foreach (FileInfo f in dirTo.GetFiles())

{

f.Delete();

}

foreach (var file in files)

{

if (FileExtenshion.Length == 0 ||

FileExtenshion.Contains(file.Extension))

{

file.CopyTo(dirTo.FullName + "\\" + file.Name);

}

}

async static public void ArchiveFile(string pathF, string targetFolder)

{

if (pathF is null)

{

throw new ArgumentNullException(nameof(pathF));

}

if (targetFolder is null)

{

throw new ArgumentNullException(nameof(targetFolder));

}

DirectoryInfo dir = new(pathF);

string path = $"{dir.FullName}//..//{dir.Name}.zip";

if(!Directory.Exists(path))

{

Console.WriteLine("\nАрхив уже создан!");

}

else

{

ZipFile.CreateFromDirectory(dir.FullName, path);

}

try

{

ZipFile.ExtractToDirectory(path, targetFolder);

}

catch(Exception ex)

{

Console.WriteLine("Файлы уже разархивированы!\n");

}

}

# 22. Запись/чтение файлов

***StreamWriter(string path, bool append, System.Text.Encoding encoding):*** параметр encoding указывает на кодировку, которая будет применяться при записи, параметр append указывает, надо ли добавлять в конец файла данные или же перезаписывать файл. Если равно true, то новые данные добавляются в конец файла. Если равно false, то файл перезаписываетсяя заново

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

{

writer.WriteLine(inf);

}

***StreamReader(string path, System.Text.Encoding encoding):*** параметр encoding задает кодировку для чтения файла

using (StreamReader reader = new(fileWithInformation))

{

string allInf = reader.ReadLine();

Console.WriteLine(allInf);

}

**FileStream**

// запись в файл

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

{

// преобразуем строку в байты

byte[] buffer = Encoding.Default.GetBytes(text);

// запись массива байтов в файл

await fstream.WriteAsync(buffer, 0, buffer.Length);

Console.WriteLine("Текст записан в файл");

}

// чтение из файла

using (FileStream fstream = File.OpenRead(path))

{

// выделяем массив для считывания данных из файла

byte[] buffer = new byte[fstream.Length];

// считываем данные

await fstream.ReadAsync(buffer, 0, buffer.Length);

// декодируем байты в строку

string textFromFile = Encoding.Default.GetString(buffer);

Console.WriteLine($"Текст из файла: {textFromFile}");

}

# 23. Время/дата

DateTime dateTime = new DateTime();

Console.WriteLine(dateTime); // 01.01.0001 0:00:00

Console.WriteLine(DateTime.Now);

DateTime date1 = new DateTime(2015, 7, 20, 18, 30, 25); // 20.07.2015 18:30:25

Console.WriteLine(date1.AddHours(3)); // 20.07.2015 21:30:25

DateTime date1 = new DateTime(2015, 7, 20, 18, 30, 25); // 20.07.2015 18:30:25

DateTime date2 = new DateTime(2015, 7, 20, 15, 30, 25); // 20.07.2015 15:30:25

Console.WriteLine(date1.Subtract(date2)); // 03:00:00

Console.WriteLine($"D: {now.ToString("D")}"); //месяц словами

DateOnly someDate = new DateOnly();

Console.WriteLine(someDate); // 01.01.0001

DateOnly now = new DateOnly(2022, 1, 6);

Console.WriteLine(now.Day); // 6

Console.WriteLine(now.DayNumber); // 738160

Console.WriteLine(now.DayOfWeek); // Thursday

Console.WriteLine(now.DayOfYear); // 6

Console.WriteLine(now.Month); // 1

Console.WriteLine(now.Year); // 2022

TimeOnly()

TimeOnly(long ticks)

TimeOnly(int hour, int minute)

TimeOnly(int hour, int minute, int second)

TimeOnly(int hour, int minute, int second, int millisecond)

TimeOnly time = TimeOnly.Parse("06:33:22");

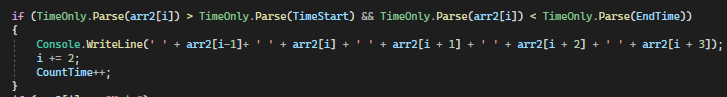
Console.WriteLine(time); // 6:33

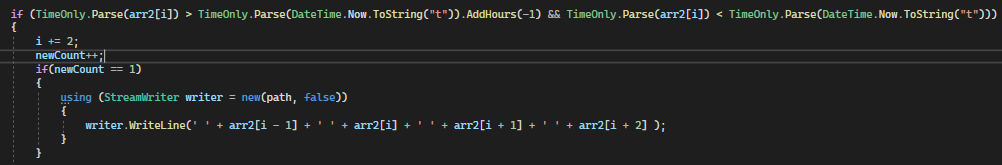
time = time.AddHours(1); // 7:33

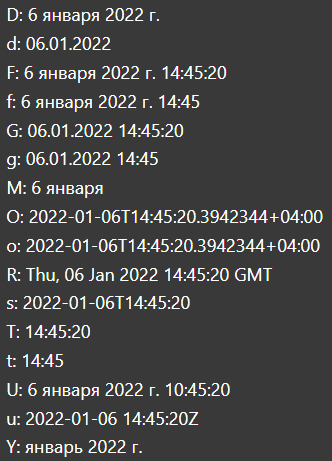
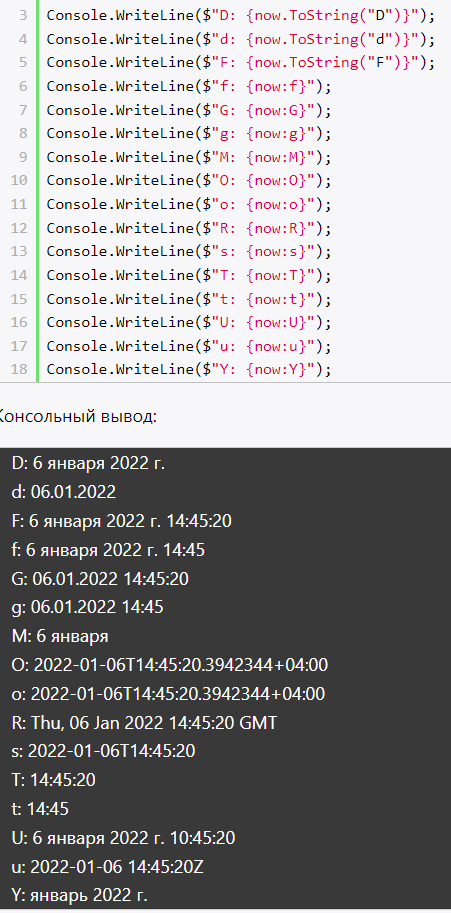
time = time.AddMinutes(-23); // 7:10

Console.WriteLine(time.ToShortTimeString()); // 7:10

Console.WriteLine(time.ToLongTimeString()); // 7:10:22







# 24. Секундомер

System.Diagnostics

Stopwatch stopwatch = new Stopwatch();

//засекаем время начала операции

stopwatch.Start();

//выполняем какую-либо операцию

**for** (**int** i = 0; i < 10001; i++)

{

Console.WriteLine(i);

}

//останавливаем счётчик

stopwatch.Stop();

//смотрим сколько миллисекунд было затрачено на выполнение

Console.WriteLine(stopwatch.ElapsedMilliseconds);

# 25. Сериализация/десериализация

string fileName = "IceAge.json";

string json = JsonConvert.SerializeObject(IceAge);

Console.WriteLine("Объект сериализован!");

File.WriteAllText(fileName, json);

Console.WriteLine("Объект десериализован:");

var data = JsonConvert.DeserializeObject<CARTOONS>(json);

Console.WriteLine("Название: " + data.\_name);

BinaryFormatter BForm = new();

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

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

{

BForm.Serialize(fs, shrek);

Console.WriteLine("Объект сериализован");

}

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

{

CARTOONS newCartoon = (CARTOONS)BForm.Deserialize(fs);

Console.WriteLine("Объект десериализован:");

Console.WriteLine(newCartoon.ToString());

}

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

SoapFormatter formatter = new();

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

{

formatter.Serialize(fs, cartoons);

Console.WriteLine("Объект сериализован");

}

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

{

CARTOONS[] newCartoon2 = (CARTOONS[])formatter.Deserialize(fs);

Console.WriteLine("Объект десериализован:");

foreach (CARTOONS p in newCartoon2)

{

Console.WriteLine("Имя: {0}", p.\_name);

}

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

XmlSerializer xmlSerializer = new (typeof(CARTOONS));

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

{

xmlSerializer.Serialize(fs, pivosaurics);

Console.WriteLine("Объект сериализован!");

}

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

{

CARTOONS? newCart = xmlSerializer.Deserialize(fs) as CARTOONS;

Console.WriteLine("Объект десериализован:");

Console.WriteLine($"Название: {newCart?.\_name} ");

}

# 26. XML

XDocument xdoc = new();

XElement C = new ("cartoon");

XAttribute NameAttr = new("name", "Tom & Jery");

XElement CountryElem = new ("country", "USA");

XElement TimeElem = new ("StartTime", "11:00");

// добавляем атрибут и элементы в первый элемент person

C.Add(NameAttr);

C.Add(CountryElem);

C.Add(TimeElem);

XElement C2 = new XElement("cartoon");

XAttribute с2NameAttr = new XAttribute("name", "Shrek");

XElement c2CountryElem = new XElement("country", "USA");

XElement c2TimeElem = new XElement("StartTime", "15:00");

C2.Add(NameAttr);

C2.Add(c2CountryElem);

C2.Add(c2TimeElem);

// создаем корневой элемент

XElement people = new ("TV\_PROGRAM");

people.Add(C);

people.Add(C2);

// добавляем корневой элемент в документ

xdoc.Add(people);

xdoc.Save("cartoon.xml");

#region requestToXml

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

XmlDocument xDoc = new();

xDoc.Load("pivosaurics.xml");

XmlElement? xRoot = xDoc.DocumentElement;

XmlNodeList? nodes = xRoot?.SelectNodes("\*");

if (nodes is not null)

{

foreach (XmlNode node in nodes)

Console.WriteLine(node.OuterXml);

}

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

XmlDocument xDoc2 = new ();

xDoc.Load("pivosaurics.xml");

XmlElement? xRoot2 = xDoc.DocumentElement;

XmlNodeList? companyNodes = xRoot?.SelectNodes("//\_name");

if (companyNodes is not null)

{

foreach (XmlNode node in companyNodes)

Console.WriteLine(node.InnerText);

}

#endregion

# 27. Потоки

var process = Process.GetCurrentProcess();

Console.WriteLine($"Текущий процесс:\nID: {process.Id} Имя: {process.ProcessName} Используемая виртуальная память: {process.VirtualMemorySize64}");

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

foreach (Process proces in Process.GetProcesses())

{

try

{

// выводим id и имя процесса

Console.WriteLine($"ID: {proces.Id} Имя: {proces.ProcessName} Приоритет: {proces.BasePriority} Время: {proces.StartTime} " +

$"Общее время использования: {proces.UserProcessorTime}");

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

public static void GetInfoDomain()

{

AppDomain domain = AppDomain.CurrentDomain;

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

Console.WriteLine($"Базовый директорий: {domain.BaseDirectory}");

Console.WriteLine($"Текущая конфигурация: {domain.SetupInformation}");

Console.WriteLine();

Console.WriteLine("Все сборки:");

Assembly[] assemblies = domain.GetAssemblies();

foreach (Assembly asm in assemblies)

{

Console.WriteLine(asm.GetName().Name);

}

}

//УСТАРЕВШЕЕ, УЖЕ НЕ ПОДДЕРЖИВАЕТСЯ((

public static void CreateDomain()

{

try

{

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

newDomain.Load(newDomain.BaseDirectory);

Console.WriteLine("Название: " + newDomain.FriendlyName);

AppDomain.Unload(newDomain);

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

}

public static void CreateThread(int n)

{

Thread myThread = new (new ParameterizedThreadStart(OutputNumber)); //ParameterizedThreadStart - значит используем параметры

myThread.Start(n);

myThread.Name = ("CreateThread");

Console.WriteLine("Имя потока: " + myThread.Name + "\t Приоритет: " + myThread.Priority + "\t Числовой идентификатор: " + myThread.ManagedThreadId

+ "\t Состояние потока: " + myThread.ThreadState);

}

public static class TwoThread

{

public static Mutex mutexObj = new();

public static void EvenNum(object n)

{

mutexObj.WaitOne(); // приостанавливаем поток до получения мьютекса

Console.WriteLine();

for (int i = 1; i <= Convert.ToInt32(n); i++)

{

if (i % 2 == 0)

{

Console.Write(i + " ");

using (StreamWriter sr = new("Ex3.txt", true))

{

sr.Write(i + " ");

}

}

}

mutexObj.ReleaseMutex(); // освобождаем мьютекс

}

public static void OddNum(object n)

{

mutexObj.WaitOne();

Thread.Sleep(500);

//using (StreamWriter sr = new("Ex3.txt", false)) ;

Console.WriteLine();

for (int i = 1; i <= Convert.ToInt32(n); i++)

{

if (i % 2 != 0)

{

Console.Write(i + " ");

using (StreamWriter sr = new("Ex3.txt", true))

{

sr.Write(i + " ");

}

}

}

mutexObj.ReleaseMutex(); // освобождаем мьютекс

}

public static void CreateThread(int n)

{

using (StreamWriter sr = new("Ex3.txt", false));

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

myThread.Priority = ThreadPriority.Highest;

Thread myThread2 = new(new ParameterizedThreadStart(OddNum));

myThread2.Priority = ThreadPriority.Lowest;

myThread.Start(n);

Thread.Sleep(500);

myThread2.Start(n);

myThread.Name = ("CreateThread\_1");

myThread2.Name = ("CreateThread\_2");

}

}

***Semaphore (int initialCount, int maximumCount):*** параметр initialCount задает начальное количество потоков, а maximumCount - максимальное количество потоков, которые имеют доступ к общим ресурсам

* WaitOne(): ожидает получения свободного места в семафоре
* Release(): освобождает место в семафоре

// запускаем пять потоков

for (int i = 1; i < 6; i++)

{

Reader reader = new Reader(i);

}

class Reader

{

// создаем семафор

static Semaphore sem = new Semaphore(3, 3);

Thread myThread;

int count = 3;// счетчик чтения

public Reader(int i)

{

myThread = new Thread(Read);

myThread.Name = $"Читатель {i}";

myThread.Start();

}

public void Read()

{

while (count > 0)

{

sem.WaitOne(); // ожидаем, когда освободиться место

Console.WriteLine($"{Thread.CurrentThread.Name} входит в библиотеку");

Console.WriteLine($"{Thread.CurrentThread.Name} читает");

Thread.Sleep(1000);

Console.WriteLine($"{Thread.CurrentThread.Name} покидает библиотеку");

sem.Release(); // освобождаем место

count--;

Thread.Sleep(1000);

}

}

}

# 28. Timer

public static void Output(object v)

{

Console.WriteLine();

Console.WriteLine("Max INT: " + int.MaxValue);

Console.WriteLine("Min INT: " + int.MinValue);

Console.WriteLine("Size INT: " + sizeof(int));

}

public static void \_timer()

{

Thread.Sleep(20);

TimerCallback tm = new TimerCallback(Output);

Timer timer = new (tm, null, 0, 100); //делегат TimerCallback - null (так как функция не принимает параметры) - start - интервал

}

# 29. Task

Stopwatch stopWatch = new Stopwatch();

Task Task1 = new(() => SearchSimpleNum.ReshetoEratosfena(n)); //все простые числа из 20

stopWatch.Start();

Task1.Start();

Task1.Wait();

stopWatch.Stop();

Thread.Sleep(1000);

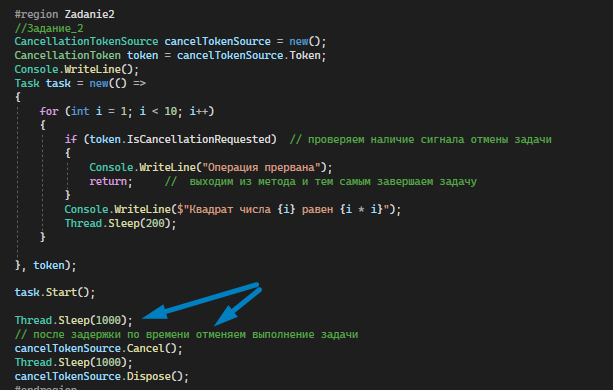
TimeSpan ts = stopWatch.Elapsed;

Console.WriteLine("Время выполнения задачи " + Task1.Id + ": " + ts);

Console.WriteLine("Идентификатор текущей задачи:" + Task1.Id + "\tСтатус задачи:" + Task1.Status + "\tЗадача завершена:" + Task1.IsCompleted);

stopWatch.Reset();

**CancellationToken:**



Task<int> task1Ex3 = new(() => Formula.\_mul(5, 2, 3));

Task<int> task2Ex3 = new(() => Formula.\_sum(2, 3, 5));

Task<int> task3Ex3 = new(() => Formula.\_pow(2, 3));

Task task4Ex3 = new(() =>

{

int result = 0;

result = task1Ex3.Result + task2Ex3.Result + task3Ex3.Result;

Console.WriteLine("\nРезультат вычислений: " + result);

});

task1Ex3.Start();

task1Ex3.Wait();

task2Ex3.Start();

task2Ex3.Wait();

task3Ex3.Start();

task3Ex3.Wait();

task4Ex3.Start();

task4Ex3.Wait();

//можно было сделать, как вложенные задачи

//var\_1

Task<int> task1Ex4 = new(() => Formula.\_sum(2, 5, 6, 10));

Task task2Ex4 = task1Ex4.ContinueWith(task =>

{

Console.WriteLine("\nСумма: " + task1Ex4.Result);

});

task1Ex4.Start();

task2Ex4.Wait();

//var\_2

Task task3Ex4 = new(() =>

{

string str1 = "\nHello";

Console.WriteLine(str1);

});

task3Ex4.Start();

task3Ex4.GetAwaiter().GetResult();

Первый параметр метода задает начальный индекс элемента в цикле, а второй параметр - конечный индекс. Третий параметр - делегат Action - указывает на метод, который будет выполняться один раз за итерацию:

Parallel.For(0, 1000000, CreateArr);

public static void CreateArr(int i)

{

arr[i] = random.Next(0,15);

}

где первый параметр представляет перебираемую коллекцию, а второй параметр - делегат, выполняющийся один раз за итерацию для каждого перебираемого элемента коллекции.

static int[] arr = new int[1000000];

Parallel.ForEach<int>(

arr,

CreateArr

);

public static void CreateArr(int i)

{

arr[i] = random.Next(0,15);

}

Parallel.Invoke(

() => Formula.\_sumOutput(10, 5, 6, 7),

() => Formula.\_mulOutput(5, 2, 3),

() => Formula.\_powOutput(3, 2)

); //задачи выполняются на разных ядрах ЭВМ

# 30. Структура main

using System.Collections.Concurrent;

using System.Diagnostics;

using System.Threading.Tasks;

…

namespace Lab\_15

{

class Program

{

static void Main(string[] argv)

{

…

# 31. Random

/Создание объекта для генерации чисел

Random rnd = **new** Random();

//Получить случайное число (в диапазоне от 0 до 10)

**int** value = rnd.Next(0, 10);

# 32. Паттерны проектирования. Паттерн Singleton.

public class App

{

public color col { get; set; }

public SizeOfText size { get; set; }

public void setColor(string colorName)

{

col = color.getInstance(colorName);

}

public void setSize(string sizeText)

{

size = SizeOfText.getInstance(sizeText);

}

}

public class color

{

private static color instance;

public string \_color { get; private set; }

protected color(string \_color)

{

this.\_color = \_color;

}

public static color getInstance(string \_color)

{

if (instance == null)

instance = new color(\_color);

return instance;

}

}

public class SizeOfText

{

private static SizeOfText instance;

public string size { get; private set; }

protected SizeOfText(string name)

{

this.size = name;

}

public static SizeOfText getInstance(string name)

{

if (instance == null)

instance = new SizeOfText(name);

return instance;

}

}

}

Main:

Console.WriteLine("\n\nЗадание 2");

App app = new();

app.setSize("12");

app.setColor("red");

Console.WriteLine("Цвет шрифта: " + app.col.\_color);

Console.WriteLine("Размер шрифта: " + app.size.size);

//попытка изменить

app.col = color.getInstance("white");

app.size = SizeOfText.getInstance("26");

Console.WriteLine("Цвет шрифта: " + app.col.\_color);

Console.WriteLine("Размер шрифта: " + app.size.size);

# 33. Паттерны проектирования. Паттерн Factory method.

abstract class Product

{ }

class ConcreteProductA : Product

{ }

class ConcreteProductB : Product

{ }

abstract class Creator

{

public abstract Product FactoryMethod();

}

class ConcreteCreatorA : Creator

{

public override Product FactoryMethod() { return new ConcreteProductA(); }

}

class ConcreteCreatorB : Creator

{

public override Product FactoryMethod() { return new ConcreteProductB(); }

}

class Program

{

static void Main(string[] args)

{

Developer dev = new PanelDeveloper("ООО КирпичСтрой");

House house2 = dev.Create();

dev = new WoodDeveloper("Частный застройщик");

House house = dev.Create();

Console.ReadLine();

}

}

// абстрактный класс строительной компании

abstract class Developer

{

public string Name { get; set; }

public Developer(string n)

{

Name = n;

}

// фабричный метод

abstract public House Create();

}

// строит панельные дома

class PanelDeveloper : Developer

{

public PanelDeveloper(string n) : base(n)

{ }

public override House Create()

{

return new PanelHouse();

}

}

// строит деревянные дома

class WoodDeveloper : Developer

{

public WoodDeveloper(string n) : base(n)

{ }

public override House Create()

{

return new WoodHouse();

}

}

abstract class House

{ }

class PanelHouse : House

{

public PanelHouse()

{

Console.WriteLine("Панельный дом построен");

}

}

class WoodHouse : House

{

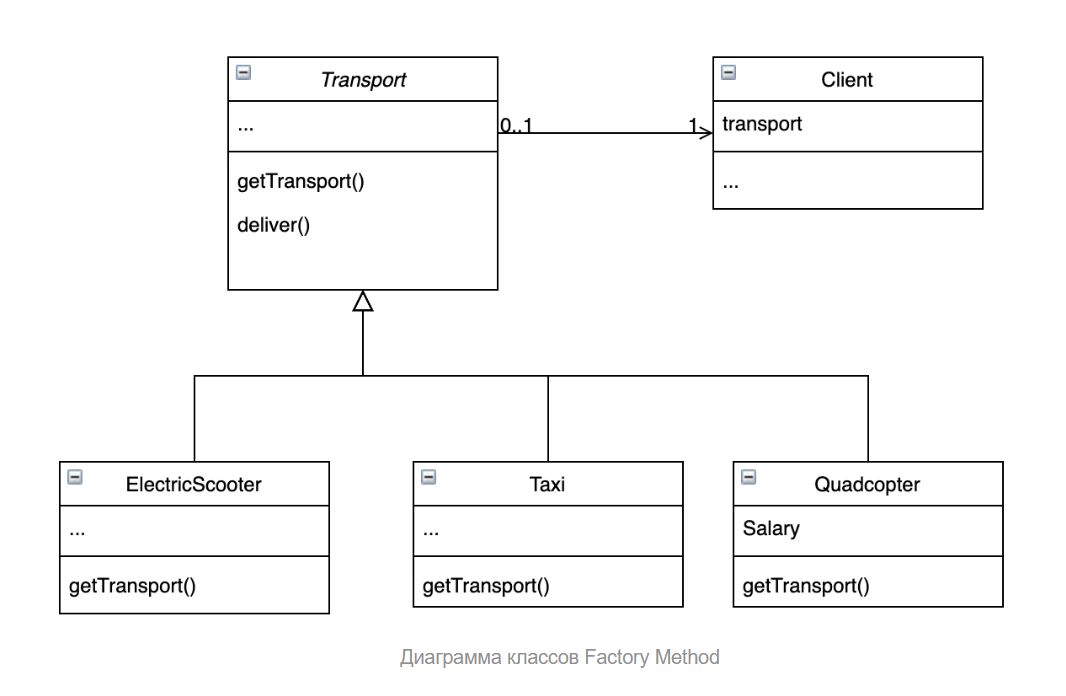
public WoodHouse()

{

Console.WriteLine("Деревянный дом построен");

}

}



# 34. Паттерны проектирования. Паттерн Abstract Factory.

abstract class AbstractFactory

{

public abstract AbstractProductA CreateProductA();

public abstract AbstractProductB CreateProductB();

}

class ConcreteFactory1 : AbstractFactory

{

public override AbstractProductA CreateProductA()

{

return new ProductA1();

}

public override AbstractProductB CreateProductB()

{

return new ProductB1();

}

}

class ConcreteFactory2 : AbstractFactory

{

public override AbstractProductA CreateProductA()

{

return new ProductA2();

}

public override AbstractProductB CreateProductB()

{

return new ProductB2();

}

}

abstract class AbstractProductA

{ }

abstract class AbstractProductB

{ }

class ProductA1 : AbstractProductA

{ }

class ProductB1 : AbstractProductB

{ }

class ProductA2 : AbstractProductA

{ }

class ProductB2 : AbstractProductB

{ }

class Client

{

private AbstractProductA abstractProductA;

private AbstractProductB abstractProductB;

public Client(AbstractFactory factory)

{

abstractProductB = factory.CreateProductB();

abstractProductA = factory.CreateProductA();

}

public void Run()

{ }

}

class Program

{

static void Main(string[] args)

{

Hero elf = new Hero(new ElfFactory());

elf.Hit();

elf.Run();

Hero voin = new Hero(new VoinFactory());

voin.Hit();

voin.Run();

Console.ReadLine();

}

}

//абстрактный класс - оружие

abstract class Weapon

{

public abstract void Hit();

}

// абстрактный класс движение

abstract class Movement

{

public abstract void Move();

}

// класс арбалет

class Arbalet : Weapon

{

public override void Hit()

{

Console.WriteLine("Стреляем из арбалета");

}

}

// класс меч

class Sword : Weapon

{

public override void Hit()

{

Console.WriteLine("Бьем мечом");

}

}

// движение полета

class FlyMovement : Movement

{

public override void Move()

{

Console.WriteLine("Летим");

}

}

// движение - бег

class RunMovement : Movement

{

public override void Move()

{

Console.WriteLine("Бежим");

}

}

// класс абстрактной фабрики

abstract class HeroFactory

{

public abstract Movement CreateMovement();

public abstract Weapon CreateWeapon();

}

// Фабрика создания летящего героя с арбалетом

class ElfFactory : HeroFactory

{

public override Movement CreateMovement()

{

return new FlyMovement();

}

public override Weapon CreateWeapon()

{

return new Arbalet();

}

}

// Фабрика создания бегущего героя с мечом

class VoinFactory : HeroFactory

{

public override Movement CreateMovement()

{

return new RunMovement();

}

public override Weapon CreateWeapon()

{

return new Sword();

}

}

// клиент - сам супергерой

class Hero

{

private Weapon weapon;

private Movement movement;

public Hero(HeroFactory factory)

{

weapon = factory.CreateWeapon();

movement = factory.CreateMovement();

}

public void Run()

{

movement.Move();

}

public void Hit()

{

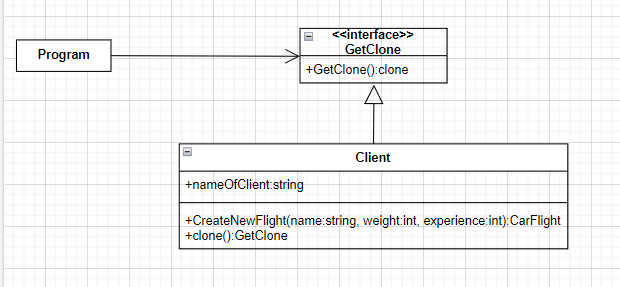
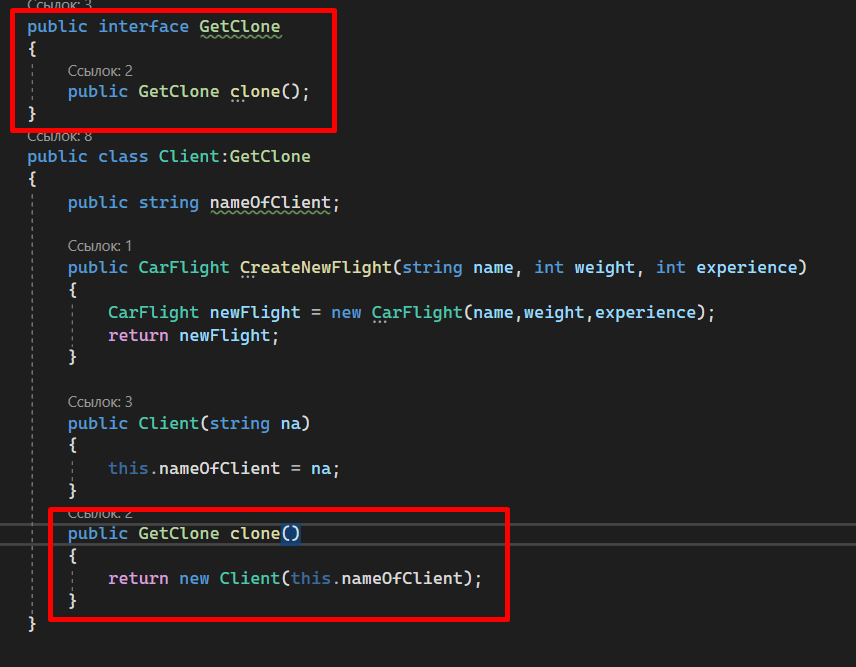
weapon.Hit();

}

}



# 35. Паттерны проектирования. Паттерн Prototype.

****

# 36. Паттерны проектирования. Паттерн Builder.

//мука

class Flour

{

// какого сорта мука

public string Sort { get; set; }

}

// соль

class Salt

{ }

// пищевые добавки

class Additives

{

public string Name { get; set; }

}

class Bread

{

// мука

public Flour Flour { get; set; }

// соль

public Salt Salt { get; set; }

// пищевые добавки

public Additives Additives { get; set; }

public override string ToString()

{

StringBuilder sb = new StringBuilder();

if (Flour != null)

sb.Append(Flour.Sort + "\n");

if (Salt != null)

sb.Append("Соль \n");

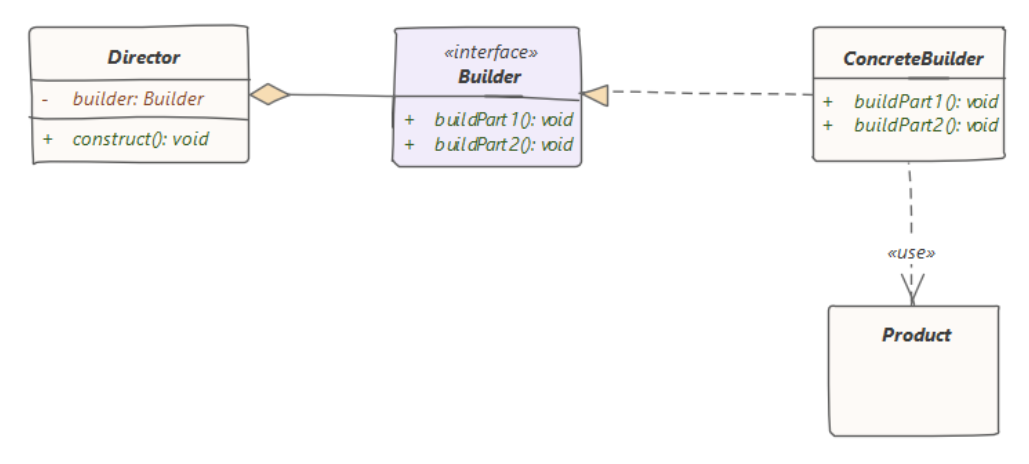
if (Additives != null)

sb.Append("Добавки: " + Additives.Name + " \n");

return sb.ToString();

}

}



# 37. Паттерны проектирования. Паттерн Adapter.

class Program

{

static void Main(string[] args)

{

// путешественник

Driver driver = new Driver();

// машина

Auto auto = new Auto();

// отправляемся в путешествие

driver.Travel(auto);

// встретились пески, надо использовать верблюда

Camel camel = new Camel();

// используем адаптер

ITransport camelTransport = new CamelToTransportAdapter(camel);

// продолжаем путь по пескам пустыни

driver.Travel(camelTransport);

Console.Read();

}

}

interface ITransport

{

void Drive();

}

// класс машины

class Auto : ITransport

{

public void Drive()

{

Console.WriteLine("Машина едет по дороге");

}

}

class Driver

{

public void Travel(ITransport transport)

{

transport.Drive();

}

}

// интерфейс животного

interface IAnimal

{

void Move();

}

// класс верблюда

class Camel : IAnimal

{

public void Move()

{

Console.WriteLine("Верблюд идет по пескам пустыни");

}

}

// Адаптер от Camel к ITransport

class CamelToTransportAdapter : ITransport

{

Camel camel;

public CamelToTransportAdapter(Camel c)

{

camel = c;

}

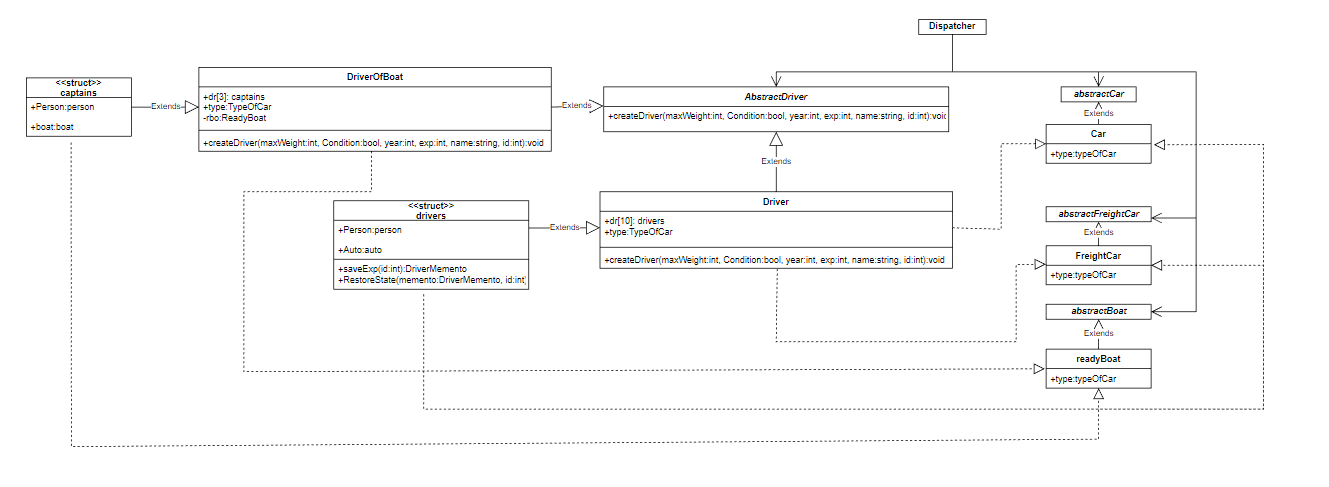
public void Drive()

{

camel.Move();

}

}



# 38. Паттерны проектирования. Паттерн Decorator.

class Program

{

static void Main(string[] args)

{

Pizza pizza1 = new ItalianPizza();

pizza1 = new TomatoPizza(pizza1); // итальянская пицца с томатами

Console.WriteLine("Название: {0}", pizza1.Name);

Console.WriteLine("Цена: {0}", pizza1.GetCost());

Pizza pizza2 = new ItalianPizza();

pizza2 = new CheesePizza(pizza2);// итальянская пиццы с сыром

Console.WriteLine("Название: {0}", pizza2.Name);

Console.WriteLine("Цена: {0}", pizza2.GetCost());

Pizza pizza3 = new BulgerianPizza();

pizza3 = new TomatoPizza(pizza3);

pizza3 = new CheesePizza(pizza3);// болгарская пиццы с томатами и сыром

Console.WriteLine("Название: {0}", pizza3.Name);

Console.WriteLine("Цена: {0}", pizza3.GetCost());

Console.ReadLine();

}

}

abstract class Pizza

{

public Pizza(string n)

{

this.Name = n;

}

public string Name { get; protected set; }

public abstract int GetCost();

}

class ItalianPizza : Pizza

{

public ItalianPizza() : base("Итальянская пицца")

{ }

public override int GetCost()

{

return 10;

}

}

class BulgerianPizza : Pizza

{

public BulgerianPizza()

: base("Болгарская пицца")

{ }

public override int GetCost()

{

return 8;

}

}

abstract class PizzaDecorator : Pizza

{

protected Pizza pizza;

public PizzaDecorator(string n, Pizza pizza) : base(n)

{

this.pizza = pizza;

}

}

class TomatoPizza : PizzaDecorator

{

public TomatoPizza(Pizza p)

: base(p.Name + ", с томатами", p)

{ }

public override int GetCost()

{

return pizza.GetCost() + 3;

}

}

class CheesePizza : PizzaDecorator

{

public CheesePizza(Pizza p)

: base(p.Name + ", с сыром", p)

{ }

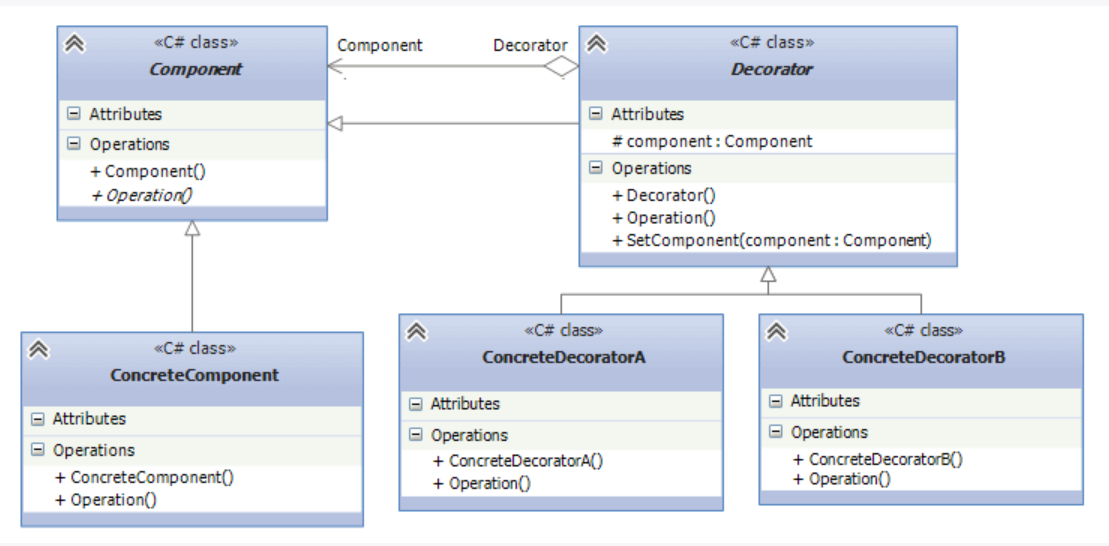
public override int GetCost()

{

return pizza.GetCost() + 5;

}

}



# 39. Паттерны проектирования. Паттерн State.

class Program

{

static void Main(string[] args)

{

Water water = new Water(WaterState.LIQUID);

water.Heat();

water.Frost();

water.Frost();

Console.Read();

}

}

enum WaterState

{

SOLID,

LIQUID,

GAS

}

class Water

{

public WaterState State { get; set; }

public Water(WaterState ws)

{

State = ws;

}

public void Heat()

{

if (State == WaterState.SOLID)

{

Console.WriteLine("Превращаем лед в жидкость");

State = WaterState.LIQUID;

}

else if (State == WaterState.LIQUID)

{

Console.WriteLine("Превращаем жидкость в пар");

State = WaterState.GAS;

}

else if (State == WaterState.GAS)

{

Console.WriteLine("Повышаем температуру водяного пара");

}

}

public void Frost()

{

if (State == WaterState.LIQUID)

{

Console.WriteLine("Превращаем жидкость в лед");

State = WaterState.SOLID;

}

else if (State == WaterState.GAS)

{

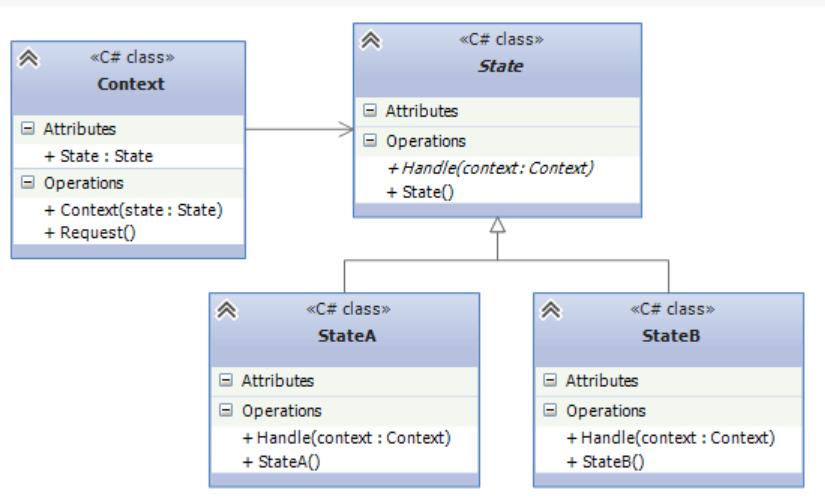
Console.WriteLine("Превращаем водяной пар в жидкость");

State = WaterState.LIQUID;

}

}

}



# 40. Паттерны проектирования. Паттерн Command.

class Program

{

static void Main(string[] args)

{

Pult pult = new Pult();

TV tv = new TV();

pult.SetCommand(new TVOnCommand(tv));

pult.PressButton();

pult.PressUndo();

Console.Read();

}

}

interface ICommand

{

void Execute();

void Undo();

}

// Receiver - Получатель

class TV

{

public void On()

{

Console.WriteLine("Телевизор включен!");

}

public void Off()

{

Console.WriteLine("Телевизор выключен...");

}

}

class TVOnCommand : ICommand

{

TV tv;

public TVOnCommand(TV tvSet)

{

tv = tvSet;

}

public void Execute()

{

tv.On();

}

public void Undo()

{

tv.Off();

}

}

// Invoker - инициатор

class Pult

{

ICommand command;

public Pult() { }

public void SetCommand(ICommand com)

{

command = com;

}

public void PressButton()

{

command.Execute();

}

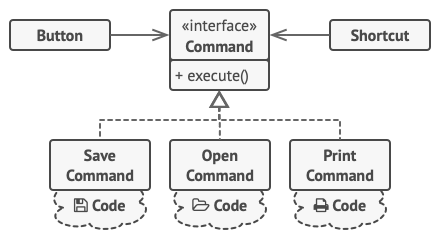
public void PressUndo()

{

command.Undo();

}

}



# 41. Паттерны проектирования. Паттерн Memento.

class Program

{

static void Main(string[] args)

{

Hero hero = new Hero();

hero.Shoot(); // делаем выстрел, осталось 9 патронов

GameHistory game = new GameHistory();

game.History.Push(hero.SaveState()); // сохраняем игру

hero.Shoot(); //делаем выстрел, осталось 8 патронов

hero.RestoreState(game.History.Pop());

hero.Shoot(); //делаем выстрел, осталось 8 патронов

Console.Read();

}

}

// Originator

class Hero

{

private int patrons = 10; // кол-во патронов

private int lives = 5; // кол-во жизней

public void Shoot()

{

if (patrons > 0)

{

patrons--;

Console.WriteLine("Производим выстрел. Осталось {0} патронов", patrons);

}

else

Console.WriteLine("Патронов больше нет");

}

// сохранение состояния

public HeroMemento SaveState()

{

Console.WriteLine("Сохранение игры. Параметры: {0} патронов, {1} жизней", patrons, lives);

return new HeroMemento(patrons, lives);

}

// восстановление состояния

public void RestoreState(HeroMemento memento)

{

this.patrons = memento.Patrons;

this.lives = memento.Lives;

Console.WriteLine("Восстановление игры. Параметры: {0} патронов, {1} жизней", patrons, lives);

}

}

// Memento

class HeroMemento

{

public int Patrons { get; private set; }

public int Lives { get; private set; }

public HeroMemento(int patrons, int lives)

{

this.Patrons = patrons;

this.Lives = lives;

}

}

// Caretaker

class GameHistory

{

public Stack<HeroMemento> History { get; private set; }

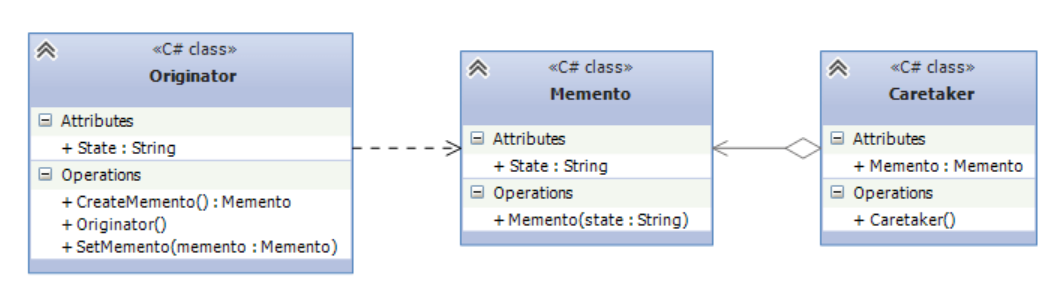
public GameHistory()

{

History = new Stack<HeroMemento>();

}

}



# 42. Паттерны проектирования. Паттерн Observer.

interface IObservable

{

void AddObserver(IObserver o);

void RemoveObserver(IObserver o);

void NotifyObservers();

}

class ConcreteObservable : IObservable

{

private List<IObserver> observers;

public ConcreteObservable()

{

observers = new List<IObserver>();

}

public void AddObserver(IObserver o)

{

observers.Add(o);

}

public void RemoveObserver(IObserver o)

{

observers.Remove(o);

}

public void NotifyObservers()

{

foreach (IObserver observer in observers)

observer.Update();

}

}

interface IObserver

{

void Update();

}

class ConcreteObserver : IObserver

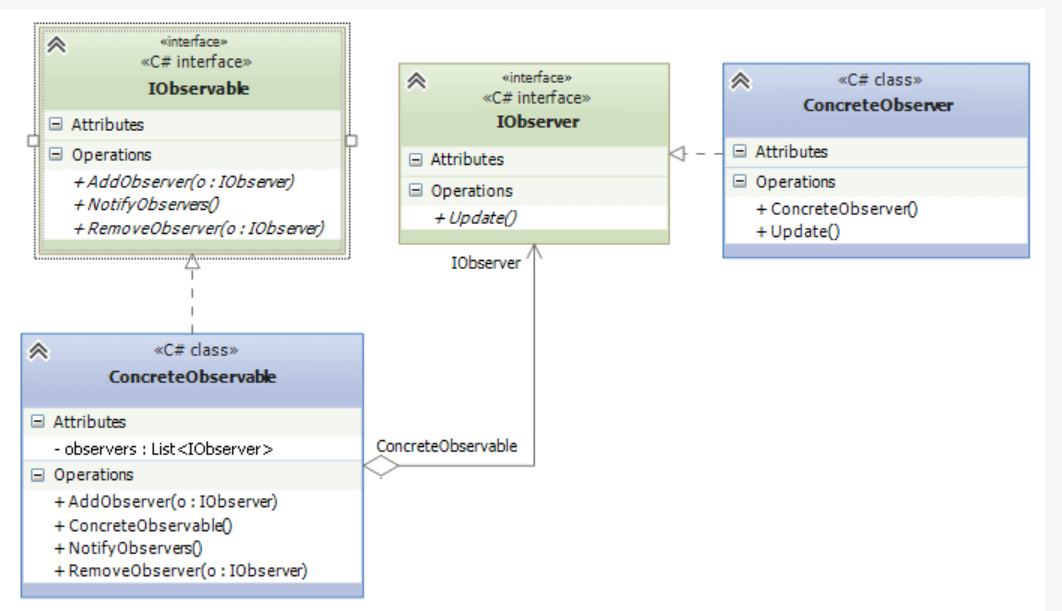
{

public void Update()

{

}

}



# 43. Паттерны проектирования. Паттерн strategy.

public interface IStrategy

{

void Algorithm();

}

public class ConcreteStrategy1 : IStrategy

{

public void Algorithm()

{ }

}

public class ConcreteStrategy2 : IStrategy

{

public void Algorithm()

{ }

}

public class Context

{

public IStrategy ContextStrategy { get; set; }

public Context(IStrategy \_strategy)

{

ContextStrategy = \_strategy;

}

public void ExecuteAlgorithm()

{

ContextStrategy.Algorithm();

}

}

[](https://radioprog.ru/uploads/media/articles/0001/06/8bb9f83b31e61fb2a3965c2f478aa755c22d5c7e.png)