**C#**

**Наследование**

*Терминология*

* Класс, который создается на основе другого класса, называется дочерним классом или производным классом. Еще его можно назвать классом наследником.
* Класс, от которого происходит наследование, называется родителем. Иногда он называется базовым классом, когда этот класс является предком для целой иерархии классов наследников.

*Синтаксис*

СпецификаторДоступа class ИмяДочернегоКласса : ИмяРодительскогоКласса

{

// поля и методы дочернего класса

}

**Наследование** - это механизм, с помощью которого можно создавать новые классы на основе уже существующих классов. При этом новые классы получают в наследство новые поля и методы того класса, на основе которого они создаются. При этом, новые классы добавляют необходимую им функциональность.

При создании объекта дочернего класса в начале вызывается конструктор родительского класса, а затем конструктор дочернего класса. При этом, по умолчанию вызывается конструктор по умолчанию родительского класса. Если в родительском классе нет конструктора по умолчанию, то попытка создания дочернего объекта приводит к ошибке компилятора. Решением этой проблемы может быть одно из следующих действий:

1. Создать в родительском классе конструктор по умолчанию.
2. В конструкторе дочернего класса явно указать вызов существующего конструктора родительского класса.

При выборе что нужно делать – наследовать или инкапсулировать нужно использовать всего лишь 2 слова: есть и иметь. Если одна сущность есть другая сущность, то нужно создавать производный класс. Если одна сущность имеет в себе другую сущность, то нужно делать инкапсуляцию.

*Замещение и переопределение методов родительского класса*

В производных классах можно создавать методы с такими же названиями и списками параметров, что и у методов родительского класса. Результатом будет замещение или переопределение. Если метод создается в дочернем классе, имеет тоже самое имя и список параметров, что и обычный метод родительского класса, то это называетя замещением. Если родительский метод является виртуальным, в таком случае произойдет переопределение виртуального метода.

*Полиморфизм*

**Полиморфизм (динамический)** – это возможность использовать одно имя для разных методов находящихся в разных классах, связанных между собой иерархией наследования. Эти методы, как правило, делают схожие действия, которые отличаются между собой предназначением класса.

**Полиморфизм (статический)** - это перегрузка функций и шаблонные классы и методы.

Для реализации динамического полиморфизма в ООП можно ссылке типа родительского класса назначать объект производного класса:

MyPoint P = new MyPoint3D(12, 50, 0);

При этом, через ссылку типа родительского класса можно вызывать только методы, описанные в родительском классе.

**Виртуальный метод** – это метод, который предназначен для вызова через ссылку родительского класса, ссылающюся на объект производного класса. То есть будет вызван метод производного класса.

**Is**

If (Ссылка is ИмяКласса)

{

// можно привести к типу

}

Else

{

// нельзя привести к типу

}

*Синтаксис*

**As**

ИмяКласса ссылка1 = Ссылка as ИмяКласса;

If (Ссылка != null)

{

// приведение возможно

}

Else

{

// приведение невозможно

}

*Синтаксис*

**Абстрактные классы**

При создании иерархии классов, в родительский класс выносятся общие поля и методы характерные для дочерних классов. Это делается с целью задать общее полиморфное поведение для дочерних классов. Часто случается, что родительский класс не может конкретезировать исполнение одного или нескольких методов, которые обязательны для дочерних классов. В этом случае можно не реализовывать такие методы в родительском классе, а объявить их абстрактными.

*Синтаксис*

СпецификаторДоступа abstract Тип ИмяМетода(СписокПараметров);

Abstrac class ИмяКласса {}

Ключевое слово abstract для метода подразумевает, что метод является виртуальным. Как только в классе появляется абстрактный метод, такой класс автоматически становиться абстрактным и также должен быть помечен ключевым словом abstract. Дочерние классы должны переопределить абстрактные методы. В противном случае дочерние классы автоматически становятся абстрактными.

**Статические поля и методы**

**Обычные поля** – поля, которые являются полями уровня объекта и содержат значения разные для каждого объекта.

**Статические поля** – это поля уровня класса. Они существуют в одном единственном экземпляре, поскольку класс в программе также существует в единственном экземпляре. Как правило, статические поля содержат общие для конкретного класса значения.

*Синтаксис*

Class SomeClass

{

СпецификаторДоступа static Тип ИмяПоля [ = value];

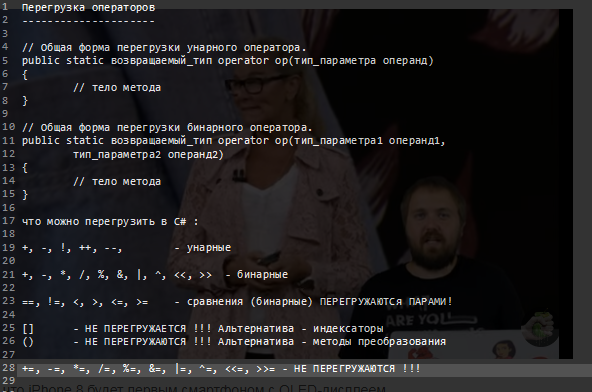
}

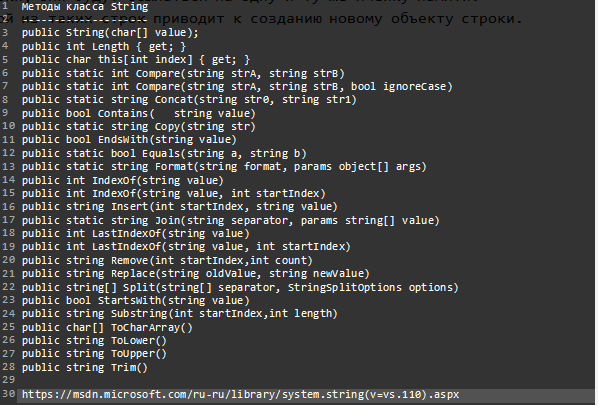
SomeClass.ИмяПоля = …;

Статические методы предназначены для работы со статическими полями. Также статические методы могут помочь решать вспомагательные задачи для объектов класса (например, создать объект класса).

**Тип данных String**

Константные строки в программе существуют только в одном экземпляре и меют 1 адресс в памяти. Если двум переменным присвоить значение одной и той же константной строки, они будут ссылаться на одну и ту же ячейку памяти. Модификация одной из таких строк приводит к созданию новому объекту строки.





**Индексаторы**

*Синтаксис*

СпецификаторДоступа ТипДанных this[ТипДанных ИмяПараметра]

{

Set

{

…

}

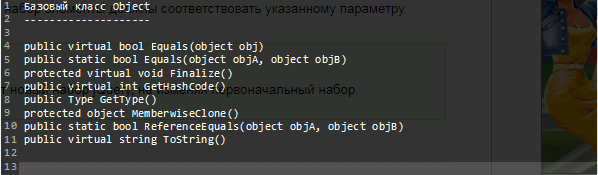
Get

{

…

}

}



Equals(object obj) – ссылается ли ссылка obj на текущий объект.

GetHashCode() – возвращает хэш-код объекта

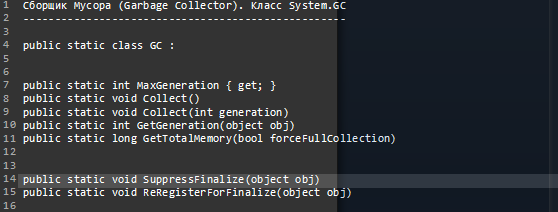
GetType() – возвращает ссылку на объект Type, в котором находится информация о текущем объекте.

MemberwiseClone() – возвращает побитовую копию объкта.

ReferenceEquals(object objA, object objB) – возвращает true, если objA и objB ссылаются на один и тот же объект.

ToString() – предназначен для преобразования объекта в строку.

Интерфейсы предназначены для задания обещго полиморфного поведения классам, не связанных между собой иерархией наследования. Интерфейсы часто сравнивают с абстрактными классами, у которых все методы абстрактны. Но более правильное определение следующее: интерфейс – это контракт, который обязуется исполнить полностью класс, взявшийся за реализацию этого интерфейса. Если класс по какой-то причине не может реализовать один или несколько методов интерфейса, он может объявить их абстрактными, тем самым самому стать абстрактным.



MaxGeneration { get; } – вернет максимальное кол-во поколений (мин. 2)

Collect() – выполняет принудительную сборку мусора

Collect(int generation) – принимает номер для которого нужно выполнить принудительную сборку мусора

GetGeneration(object obj) – возвращает номер поколения, к которому причислен объект

GetTotalMemory – возвращается кол-во использованой памяти на данный момент в байтах

SuppressFinalize(object obj) – сообщает платформе .NET, что для данного объекта не нужно вызывать деструктор

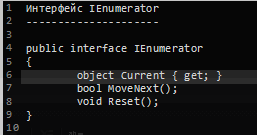
ReRegisterForFinalize(object obj) – отменяет действие предудыщего метода

**Коллекции .NET**

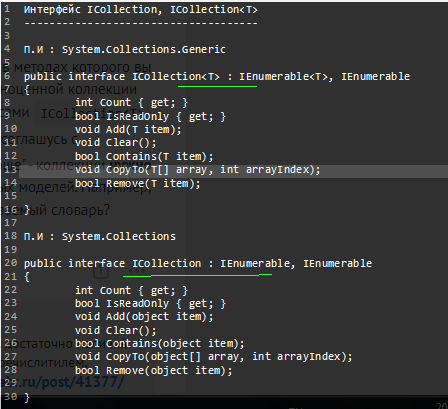
**Коллекция** – это структура данных предназначенная для хранения данных одного типа, которая работает по определенным правилам и меняет свой размер в процессе работы программы таким образом, чтобы вместить в себя все размещенные значения.

*Интерфейсы коллекций*

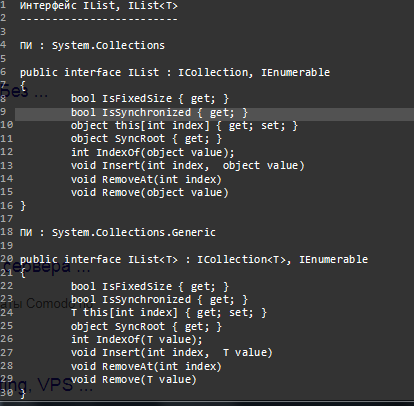
* IEnumerable – описывает методы для перебора коллекций с помощью конструкции foreach

**

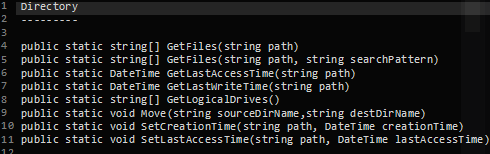
* Icollection

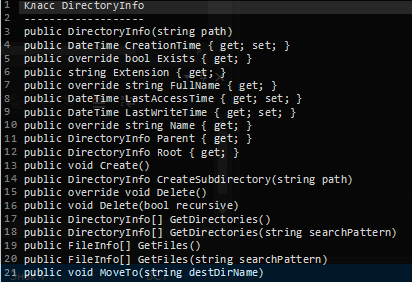
**

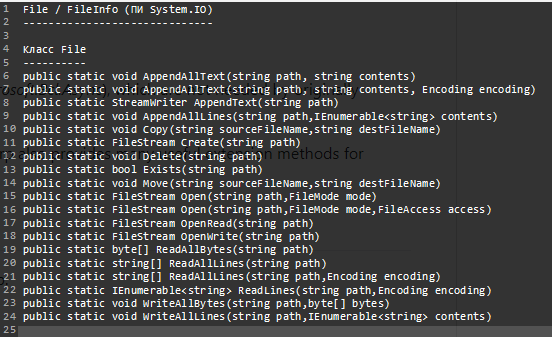
* Ilist

**

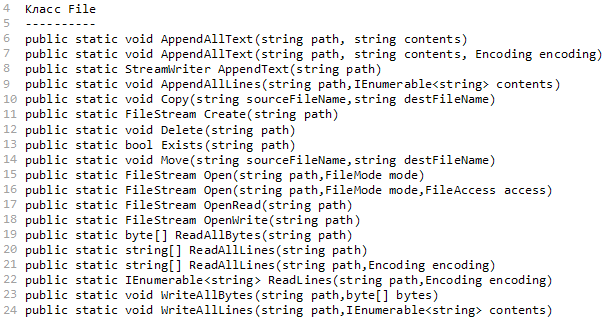
Directories







**Class File**



6) дописывает в конец файла текст (сам закрывает файл)

File.AppendAllText("C:/Users/.../test.txt", "Hello World!!!");

7) то что и первое но с указанием кодировки (сам закрывает файл)

File.AppendAllText("C:/Users/09220/Desktop/test.txt", "Hello World!!!\r\n Привет мир!!!", Encoding.UTF8);

8) открывает файл для дозаписи

//дозапись с открытием файла и закрытием

StreamWriter SW = File.AppendText("C:/Users/09220/Desktop/test.txt");

SW.Write("Microsoft Windows Forever!!!");

SW.Close();

9) добавляет в конец файла коллекцию строк

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

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

col.Add("Строка" + i);

File.AppendAllLines(@"C:\Users\09220\Desktop\abc.txt", col); //если нет файла то файл создаётся

10) копирует файл

File.Copy(@"C:\Users\09220\Desktop\abc.txt", @"C:\Users\09220\Desktop\TestDir\abc.txt");

11) созд файл по указаному пути и возвр ссылку на объект string файловый поток байтовый с помощью которого можно записывать

поторк ввода - это послед байт предназ для чтения

поторк ввывода - это посл байт для записи

потоки ввода-вывода могут быть файловые в памяти и сетивые

FileStream FS = File.Create(@"C:\Users\09220\Desktop\csharp.txt");

byte[] buf = { 65, 66, 67, 68, 69, 70, 71, 72, 56, 54, 53 };

FS.Write(buf, 0, buf.Length); //запись массива байт в файл

FS.Close();

12) удаляет файл по указаному пути

13) существует ли файл по указаному пути

14) переносит файл

15) открывает файл (путь/режим открытия файла)

возв ссылочку на файл поток

**Режимы открытия файла:**

Append - открыть для дозаписи (запись в конец)

Create - создать (если существует то удалить всё содержимое)

CreateNew - создать файл если не существует (если есть то искл сит IOExceptio).

Open - открыть сущ файл (если нет то исключ сит)

OpenOrCreate - открыть сущ файл если нет файла то создать

Truncate - удалить всё содер в фаайле

====

FileStream FS = File.Open("C:/...../1.txt", FileMode.Open | FileMode.Truncate);

===

FileStream FS1 = File.Open(@"C:\Users\09220\Desktop\csharp.txt", FileMode.Open);

byte[] buf1 = new byte[1024];

int cnt = FS1.Read(buf1, 0/\*от\*/, buf1.Length/\*до\*/);

FS1.Close();

Console.WriteLine("Прочитано байт : {0}", cnt);

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

Console.WriteLine("buf1[{0}] = {1}", i, buf1[i]);

16)

**Перечень FileAcces**

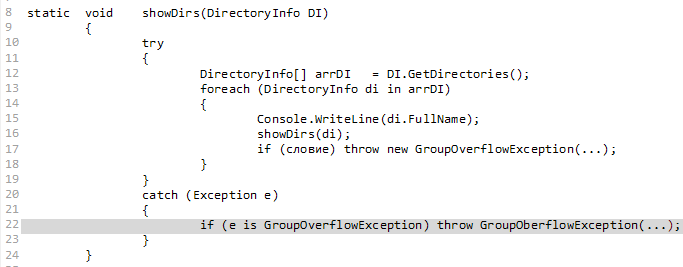
Read - только для чтения

Write - только для записи

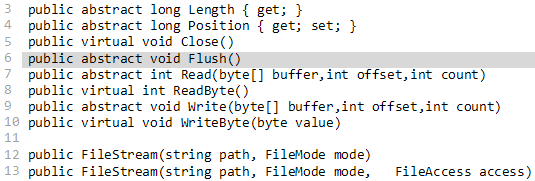
ReadWrite - чтение - запись

FileStream FS1 = File.Open(@"C:\Users\09220\Desktop\csharp.txt", FileMode.Open, **FileAccess.Read**);

dicshionary не позв добавлять ключи объект которого уже есть



**Класс FileStream**

****

3) вернёт длину потока (файла)

4) позволит установит поз в файле от куда читать или записывать (в байтах)

5) закрывает байтовый поток, при этом если ост не дозаписанные байты то они будут записаны в файл

6) Любой поток ввода - вывода (записать содерж внутреннего буфера в файл (принудительно (как автосохранение)))

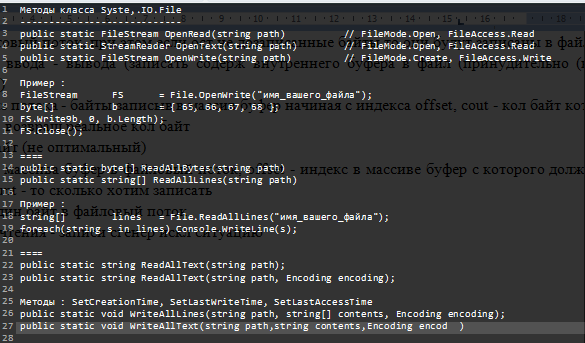
7) читает байты из потока - байты записыв в массив буфер начиная с индекса offset, cout - кол байт котор хотим прочитать. возвращ реальное кол байт

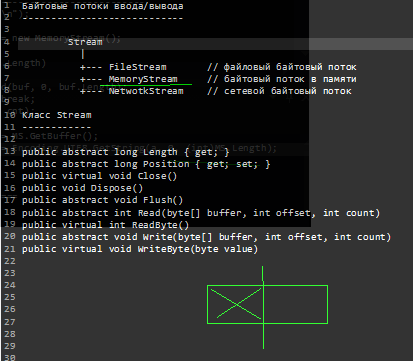
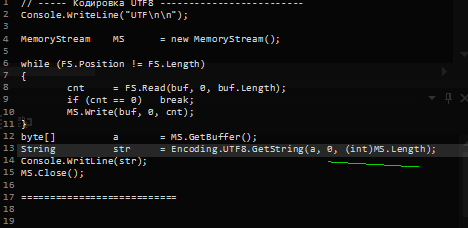
8) Читает один байт (не оптимальный)

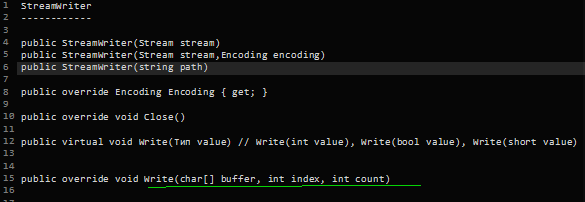
9) записыв содер массива буфер в файловый поток, offset - индекс в массиве буфер с которого должны записываться, count - то сколько хотим записать

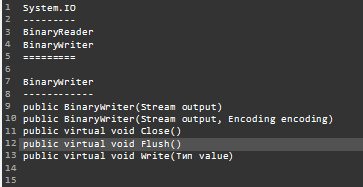
10) записывает один байт в файловый поток

в случае ошибок чтения - записи сгенер искл ситуацию









**Делегаты**

Делегаты предназначены для реализации механизмо обратного вызова. Делегаты в .NET являются типом. Делегаты объявляются при помощи ключевого слова **delegate**.

*Синтаксис*

СпецификаторДоступа delegate Тип ИмяДелегата(Cписок параметров);

Синхронный вызов – вызов метода в текущем потоке исполнения, при этом поток не будет исполняться дальше, пока метод не завершит свою работу.

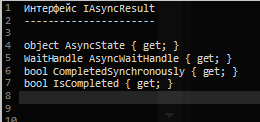
Асинхронный вызов означает, что метод вызывается во вторичном потоке, при этом текущий поток продолжает выполняться, не дожидаясь завершения работы метода во вторичном потоке.

BeginInvoke(params, IasyncCallback callback, object obj) – начинает исполнение метода во вторичном потоке. Когда метод во вторичном потоке завершит свою работу , то будет вызван метод EndInvoke();. Возвращает ссылку на объект, реализующий интерфейс и AsyncResult; Эту ссылку необходимо сохранить, чтобы передать ее в метод EndInvoke для получения результата.

* Callback – делегат, ссылающийся на метод, который будет вызван по завершению работы метода исполняющегося во вторичном потоке, этот параметр может быть null.
* Obj – ссылка на объект, который будет передан в метод callback

Callback и obj предназначены для автоматического получения результата работы вторичного потока

Для вызова метода во вторичном потоке, делегат должен ссылаться ТОЛЬКО на один метод, а не на несколько.



* AsyncState – ссылка на объект, содаржащем информацию об осинхронном вызове
* AsyncWaitHandle - Ссылка на объект синхронизации, которую можно использовать для приостанвоки работы потока до тех пор, пока метод, на который ссылается делегат(использующийся во вторичном потоке) не прекратит работу.
* CompletedSynchonously – свойство, которое вернет признак, завершился метод синхронно или асинхронно
* IsCompleted – свойство, которое вернет true, если метод, на который ссылается делегат завершился.

**AsyncCallback**

AsyncCallback - предназначен для вызова методов, которые были запущены во вторичном потоке, с помощью BeginInvoke(). Этот метод будет запущен автоматически для получения результата исполнения метода, запущенного с помощью BeginInvoke().

