# 一 C#Syntax 基本结构

1 using System;

2

3 namespace HelloWorld

4 {

5 class Program

6 {

7 static void Main(string[] args)

8 {

9 Console.WriteLine("Hello World!");

10 }

11 }

12 }

**Line 1:** using System means that we can use classes from the System namespace.

**Line 2:** A blank line. C# ignores white space. However, multiple lines makes the code more readable.

**Line 3:** namespace is used to organize your code, and it is a container for classes and other namespaces.类和其他域名的组织框架

**Line 4:** The curly braces {} marks the beginning and the end of a block of code.

**Line 5:** class is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class Program. 每条指令必须包含在一个类里

**Line 7:** Another thing that always appear in a C# program, is the Main method. Any code inside its curly brackets {} will be executed. You don't have to understand the keywords before and after Main. You will get to know them bit by bit while reading this tutorial.

**Line 9:** Console is a class of the System namespace, which has a WriteLine() method that is used to output/print text. In our example it will output "Hello World!".

If you omit the using System line, you would have to write System.Console.WriteLine() to print/output text.

Console是System里包的域名,包含很多函数(WriteLine)

**Note:** Every C# statement ends with a semicolon **;**.

**Note:** C# is case-sensitive: "MyClass" and "myclass" has different meaning.

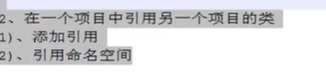
**Note:** Unlike [Java](https://www.w3schools.com/java/default.asp), the name of the C# file does not have to match the class name, but they often do (for better organization). When saving the file, save it using a proper name and add ".cs" to the end of the filename. To run the example above on your computer, make sure that C# is properly installed: Go to the [Get Started Chapter](https://www.w3schools.com/cs/cs_getstarted.php) for how to install C#. The output should be:



### 命名空间

直接写最上面

在一个项目中引用另一个项目的类



## C#Output

To output values or print text in C#, you can use the WriteLine() method:

* Console.WriteLine(„Hello World!“); 输出字符串,带引号““
* Console.WriteLine(3+3); 输出计算结果

## C#Comments

* //

// This is a comment

Console.WriteLine("Hello World!");

* Any text between /\* and \*/ will be ignored by C#.

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

Console.WriteLine("Hello World!");

## C# Variables变量

In C#, there are different **types** of variables (defined with different keywords), for example:

* int - stores integers (whole numbers整数), without decimals, such as 123 or -123
* double - stores floating point numbers浮点小数, with decimals, such as 19.99 or -19.99
* decimal – 128位数据类型.同double相比,decimal具有更高的精度和更小的范围存储金钱,值后边需要加上m,
* char - stores single characters单字符, such as 'a' or 'B'. Char values are surrounded by single quotes
* string - stores text字符串, such as "Hello World". String values are surrounded by double quotes
* bool - stores values with two states真假值: true or false

### 定义变量

**rule**

1. create a name
2. give variable a value
3. use variable

string name = "John";

int myNum = 15;

int myNum;

myNum = 15;

重赋值

int myNum = 15;

myNum = 20; // myNum is now 20

其他

int myNum = 5;

double myDoubleNum = 5.99D;

char myLetter = 'D';

bool myBool = true;

string myText = "Hello";

绿色波浪线是警告线

红色波浪线是报错线

### 定义常量const

**const** int myNum = 15;

myNum = 20; // error

The const keyword is useful when you want a variable to always store the same value常量不会被改变, so that others (or yourself) won't mess up your code. An example that is often referred to as a constant, is PI (3.14159...).

**Note:** You cannot declare a constant variable without assigning the value. If you do, an error will occur: A const field requires a value to be provided. **常量必须被赋值**

### Display Variables变量运算

字符串加,没有减

string name = "John";

Console.WriteLine("Hello " + name);

string firstName = "John ";

string lastName = "Doe";

string fullName = firstName + lastName;

Console.WriteLine(fullName);

算术加减

int x = 5;

int y = 6;

Console.WriteLine(x + y); // Print the value of x + y

## C# Identifiers定义变量名字的规则

* Names can contain letters, digits and the underscore character (\_)
* Names must begin with a letter
* Names should start with a lowercase letter and it cannot contain whitespace

(camel命名法:首个单词首字母小写,其他单词首字母大写----highSchool或者highSchoolStudent;

Pascal命名规范:每个单词首字母都大写.多给类或者函数命名------MakeCake

全大写命名规则:只限于缩写DHL)

* Names are case sensitive ("myVar" and "myvar" are different variables)(HTML语言不区分大小写)
* Reserved words (like C# keywords, such as int or double) cannot be used as names

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Size** | **Description** |
| int | 4 bytes | Stores whole numbers from -2,147,483,648 to 2,147,483,647 |
| long | 8 bytes | Stores whole numbers from -9,223,372,036,854,775,808 to  9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal  digits |
| bool | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter, surrounded by single quotes |
| string | 2 bytes per character | Stores a sequence of characters, surrounded by double quotes |

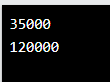
A floating point number can also be a scientific number with an "e" to indicate the power of 10:

float f1 = 35e3F;

double d1 = 12E4D;

Console.WriteLine(f1);

Console.WriteLine(d1);

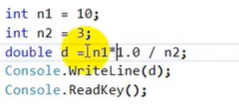
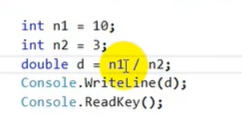


## C# Type Casting 变量类型转换

**显式类型转换和隐式类型转换**

两种类型必须兼容才能转换!

* 隐式类型转换:两种类型兼容,小转大
* 强制类型转换:两种类型兼容,大转小

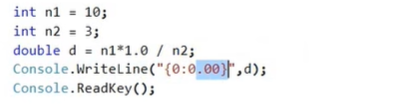


结果:3.33333

只要把其中一个的类型转为double,那么整个表达式都是double类型

结果:3

两个数都是int,那么结果也是int



结果:3.33

使用占位符输出,并限制输出的位数,就可以控制double的输出结果

Type casting is when you assign a value of one data type to another type.

In C#, there are two types of casting:

* **Implicit Casting由小到大** (automatically) - converting a smaller type to a larger type size  
  char -> int -> long -> float -> double
* **Explicit Casting由大到小(需手动)** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char

自动

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

Console.WriteLine(myInt); // Outputs 9

Console.WriteLine(myDouble); // Outputs 9

手动

double myDouble = 9.78;

int myInt = (int) myDouble; // Manual casting: double to int

Console.WriteLine(myDouble); // Outputs 9.78

Console.WriteLine(myInt); // Outputs 9

Convert函数>两种不同类型的转换函数

It is also possible to convert data types explicitly by using built-in methods, such as Convert.ToBoolean,

Convert.ToDouble,

Convert.ToString,

Convert.ToInt32 (int) and Convert.ToInt64 (long):

### String 转double int

1.int number = int.Parse(“123abc”)

2.练习operator-2.2.1

[Convert String to Double in C# | How to Convert String to Double in C#? (educba.com)](https://www.educba.com/convert-string-to-double-in-c-sharp/)

using System.Globalization;

NumberFormatInfo provider = new NumberFormatInfo();

provider.NumberDecimalSeparator = ".";

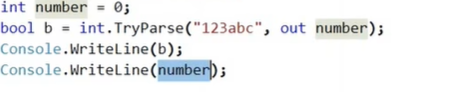
provider.NumberGroupSeparator = ",";

double doubleVal = Convert.ToDouble("855.65", provider);

1. int.TryParse( **i**, **out** **j**)

i = 要转换的字符串

j = 把转换成功的值赋给j



试着把字符串转换为int ,如果转换成功b=TRUE,number=数字;

不成功b=FALSE,number=0;

Out表示把指针直到变量上,将变量值改变

## C# User Input写入功能

Console.WriteLine() is used to output (print) values

* Console.ReadLine() to get user input

// Type your username and press enter

Console.WriteLine("Enter username:");

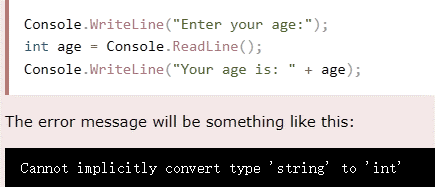
// Create a string variable and get user input from the keyboard and store it in the variable

string userName = Console.ReadLine();

// Print the value of the variable (userName), which will display the input value

Console.WriteLine("Username is: " + userName);

* Console.ReadLine() 返回值是字符串,输出的也是字符串The Console.ReadLine() method returns a string. Therefore, you cannot get information from another data type, such as int.



正确的 使用Convert.ToInt32()

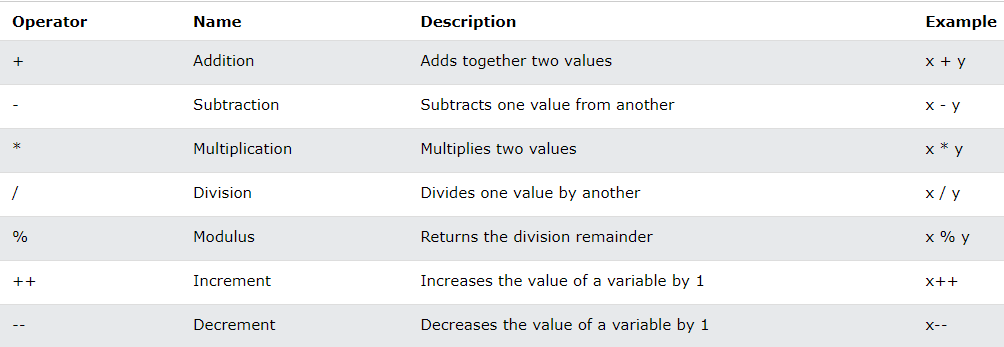
Console.WriteLine("Enter your age:");

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

Console.WriteLine("Your age is: " + age);

## C# Operators

### Arithmetic算术运算子



**一元运算符优先级大于二元**

自加,自减

余数

**详解:**

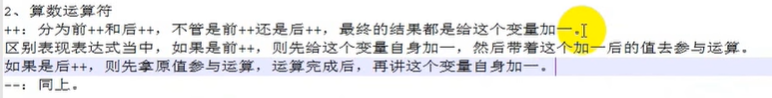
int a = 5;

int b = a++ + ++a \* 2 + --a + a++;

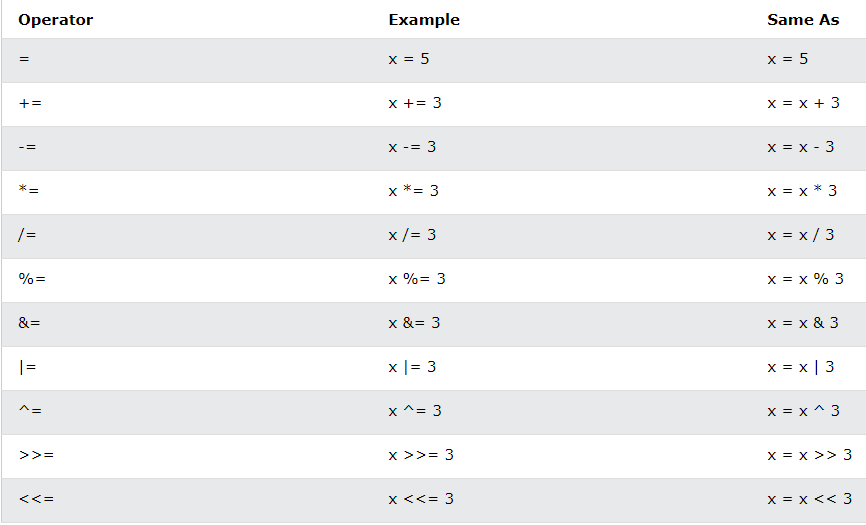
Console.WriteLine(a);

Console.WriteLine(b);

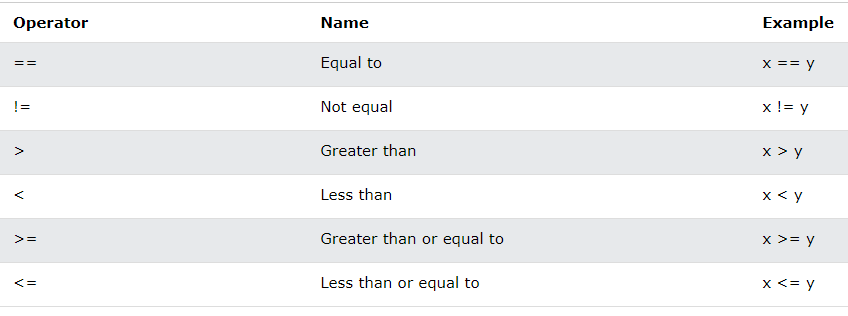




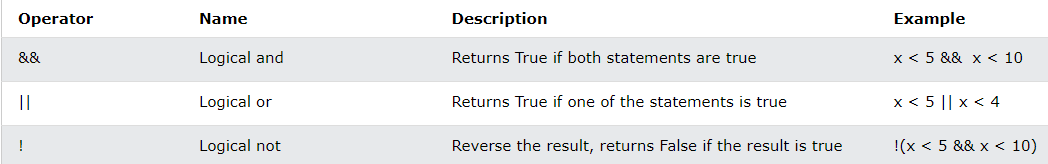
### Assignment



### Comparison



### Logical Operators 逻辑子/ 与或非



逻辑与的优先级大于逻辑或

与&&和&的区别:

5＜3 && 5＞3 当前一个条件false**就不**执行后边的语句了

5＜3 & 5＞3 当前一个条件false **还**  执行后边的语句

或||和|的区别:

同理: 当前一个条件成是true也就不用执行后边一条了

**双逻辑子省空间**

## C# Math Math函数

Math.Max(x,y);

Math.Min(x,y);

Math.Sqrt(x);幂

Math.Abs(x);绝对值

Math.Round(x);取接近整数

## C# Strings

### Strings

* 取长

string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

Console.WriteLine("The length of the txt string is: " + txt.Length);

* 变大小写

string txt = "Hello World";

Console.WriteLine(txt.ToUpper()); // Outputs "HELLO WORLD"

Console.WriteLine(txt.ToLower()); // Outputs "hello world"

### Concatenation String.Concat(x,y)

string firstName = "John ";

string lastName = "Doe";

string name = string.Concat(firstName, lastName);

Console.WriteLine(name);

WARNING!

C# uses the + operator for both addition and concatenation.

**Remember:** Numbers are added. Strings are concatenated.

### String Interpolation 定量和变量放在一起

Another option of [string concatenation](https://www.w3schools.com/cs/cs_strings_concat.php), is **string interpolation**, which substitutes values of variables into placeholders in a string. Note that you do not have to worry about spaces.用美元符$可以把定量和变量放在一起””,变量用{}括起来:

string firstName = "John";

string lastName = "Doe";

string name = $"My full name is: {firstName} {lastName}";

Console.WriteLine(name);

### Access Strings对字符串定位

* 取固定位字符 []

string myString = "Hello";

Console.WriteLine(myString[0]); // Outputs "H"

字符串以0位为起始位

* 输出字符位置IndexOf( )

string myString = "Hello";

Console.WriteLine(myString.IndexOf("e")); // Outputs "1"

* 从固定位置开始输出 Substring( )

starting from the specified character position/index, and returns a new string. This method is often used together with IndexOf() to get the specific character position常和IndexOf()一起用:

// Full name

string name = "John Doe";

// Location of the letter D

int charPos = name.IndexOf("D");

// Get last name

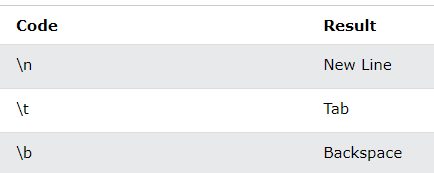
string lastName = name.Substring(charPos);

// Print the result

Console.WriteLine(lastName);

### Special Characters 转义字符





在Windows里应该用\r\n

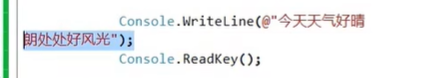
\b表示一个退格键,前面的字符消失,

\b放到字符串两边没有效果

在定义路径的时候应该给路径都用两个\\或者使用@

**@** : 表示不转译 和 保留原格式输出

string path = @”F:\\leben\music\a\b\v\c\jayjoy.mp3”;





## C# Booleans

赋值:  true or false

输出:True or False

## 语法结构大框(图)



## C# If ... Else

凡是能用ifelse做的都能用三元表达式做

简写

int time = 20;

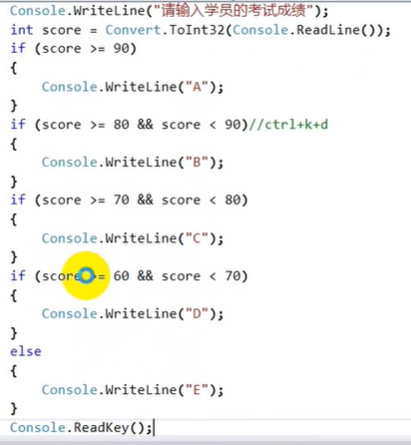
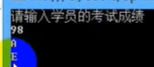
string result = (time < 18) ? "Good day." : "Good evening.";

Console.WriteLine(result);

### else就近原则:

else只和离它最近的if配对,只要那个if不符和,马上执行else.

**错误代码:**

应该把这里的else改成if

很多个if

## C# Switch检索

The break and default keywords will be described later in this chapter.

接触到break会立即退出;没有检索到结果用default

也可以用做范围检索:想办法把范围变成定值

Int expression = xxx;

switch(expression)

{

case x:

// code block

break;

case y:

// code block

break;

default:

// code block

break;

}

## C# While Loop

* while 先判断再运行

int i = 0;

while (i < 5)

{

Console.WriteLine(i);

i++;

}

* do while 适合至少运行一次的操作

int i = 0;  
do

{

Console.WriteLine(i);

i++;

}

while (i < 5);

### berak的用法

一般与if连用

作用>跳出当前循环

## C# For Loop

For循环适合已知循环次数的操作.When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:

打完for按两下tap键就出来了

逆向for循环 forr + 两下tap

for (*statement 1*; *statement 2*; *statement 3*)

{

*// code block to be executed*

}

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

### For循环和while循环的区别???

For循环适用于已知循环次数(循环次数一定是有限的,知道到哪一步跳出去);当for(;;)时,等同于while().

While()只判断括号内的值是否为真就好,可以一直循环



### The foreach Loop遍历函数

一般被用来遍历数组

There is also a foreach loop, which is used exclusively to loop through elements in an **array**

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

foreach (string i in cars)

{

Console.WriteLine(i);

}



## C# Break and Continue

break立即跳出当前循环

The break statement can also be used to jump out of a **loop**.

Continue 跳出当前小循环,例:不会输出4

int i = 0;

while (i < 10)

{

if (i == 4)

{

i++;

continue;

}

Console.WriteLine(i);

i++;

}

## C# Arrays

### 创建数组

* 定义 : new String[ ]

// Create an array of four elements, and add values later只定义长度不赋值

string[] cars = new string[4];

int[] myNum = {10, 20, 30, 40};

// Create an array of four elements and add values right away 定义长度且赋值

string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements without specifying the size 赋值不定义长度

string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

Difference?

// Create an array of four elements, omitting the new keyword, and without specifying the size

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

* 更改其中数据

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

Console.WriteLine(cars[0]);

// Now outputs Opel instead of Volvo

* 得到长度

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

Console.WriteLine(cars.Length);

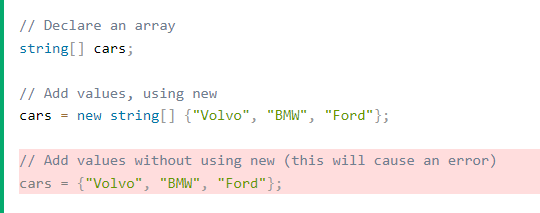
// Outputs 4

* 注意

It is up to you which option you choose. In our tutorial, we will often use the last option, as it is faster and easier to read.

However, you should note that if you declare an array and initialize it later, you have to use the new keyword:

只定义未赋值,这种情况下,当想赋值时必须用new



### 遍历数组 for / foreach

* for

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

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

{

Console.WriteLine(cars[i]);

}

* foreach (better)

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

foreach (string i in cars)

{

Console.WriteLine(i);

}

* **for each** string element (called **i** - as in **i**ndex) in **cars**, print out the value of**i**
* foreach更简单易读

### Sort an Array (整理)

* .Sort可以排列字符串数组String[ ]或者数字数组int[ ]

// Sort a string

string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

Array.Sort(cars);

foreach (string i in cars)

{

Console.WriteLine(i);

}

// Sort an int

int[] myNumbers = {5, 1, 8, 9};

Array.Sort(myNumbers);

foreach (int i in myNumbers)

{

Console.WriteLine(i);

}

* 挑选最大最小 string and int

用来挑选最大最小的函数功能在System.Linq包里

using System;

using System.Linq;

namespace MyApplication

{

class Program

{

static void Main(string[] args)

{

int[] myNumbers = {5, 1, 8, 9};

Console.WriteLine(myNumbers.Max()); // returns the largest value

Console.WriteLine(myNumbers.Min()); // returns the smallest value

Console.WriteLine(myNumbers.Sum()); // returns the sum of elements

}

}

}

**Array.Reverse(nums);**

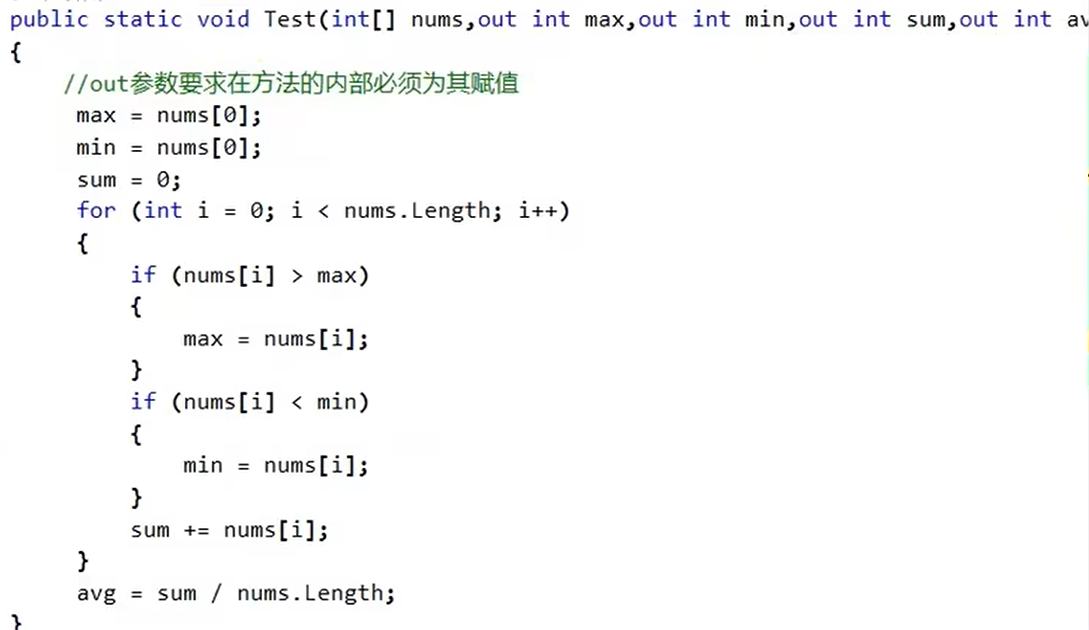
倒序排列(不是从小到大排).与正序排列相对.

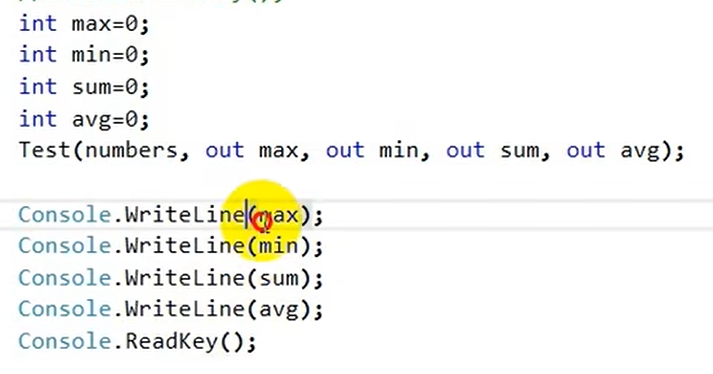
### out参数

* 如果一个方法中,返回多个相同类型的值的时候,可以考虑返回一个数组.
* 但是,如果返回多个不同类型的数组,返回数组就不行了,那么这个时候,我们可以考虑一下out使用参数

**out参数要求在方法的内部必须为其赋值**

**使用时out 参数必须声明类型,但无需赋值.**

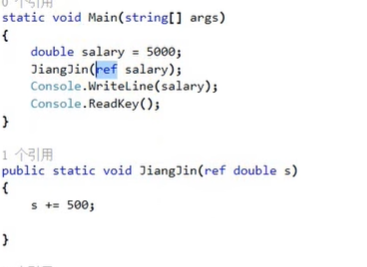




### ref参数 指针?

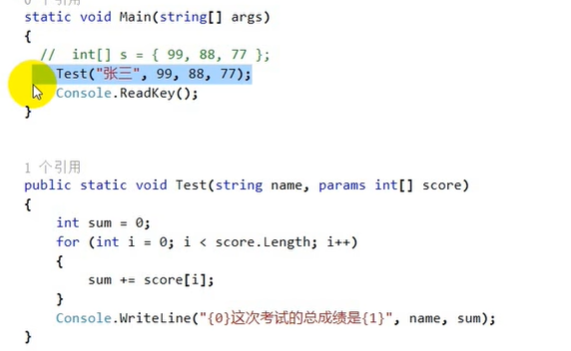
能够将一个变量带入一个方法中进行法改变,改变完成后,再将改变后的值带出方法.

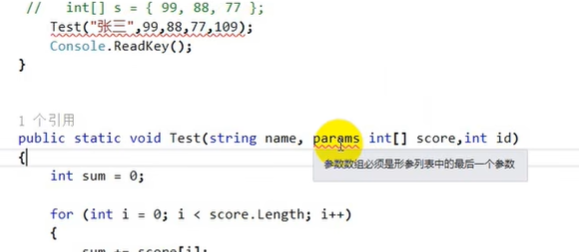
Ref参数要求在方法外必须为其赋值,而方法内可以不赋值.



### params可变参数

* 将实参列表中跟可变参数数组类型一致的元素都当做数组的元素去处理
* 参数数组必须是形参列表的最后一个元素

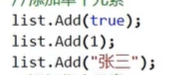




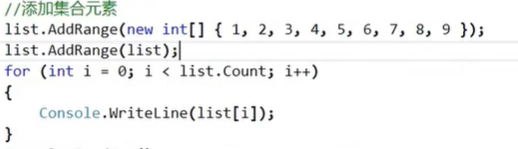
## C#集合

### C#ArrayList集合的方法

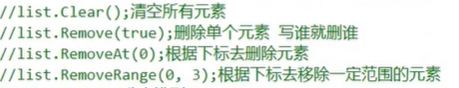
1. 添加单个元素



1. 添加集合元素AddRange,可以添加自己



1. 移除元素



1. 升序排列,反转排列



1. 插入单个元素,插入集合元素
2. 判断是否包含某个指定的元素



* 集合长度的问题

每次集合中实际包含的元素个数(count)超过了可以包含的元素的个数(capcity)的时候,集合就会向内存中申请多开辟一倍的空间,来保证集合长度一直够用

### List泛型集合

创建泛型集合对象



一定要加括号!!

泛型集合可以转换为数组,int或者string



数组转集合

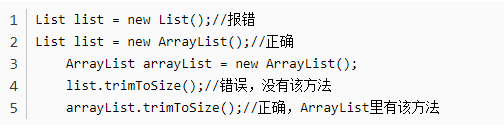


* List 和ArrayList 的区别
  + List是一个接口，而ArrayList是List接口的一个实现类
  + ArrayList类继承并实现了List接口

因此，List接口不能创建实例对象，但是可以为List接口创建一个指向自己的对象引用。而ArrayList实现类的实例对象就在这充当指向List接口的对象引用。



这体现了面向对象的多态性



它就是一个List对象，而有些ArrayList类具有的，但是List接口没有的属性和方法，就不能再用了。 而ArrayList list=new ArrayList();

创建对象则保留了ArrayList的所有属性和方法。

**问题的关键：为什么要用List list = new ArrayList(),而不用ArrayList alist = new ArrayList()呢?**

问题就在于List接口有多个实现类，现在你用的是ArrayList，也许哪一天你需要换成其它的实现类，如LinkedList或者Vector等等，这时你只要改变这一行就行了： List list = new LinkedList(); 其它使用了list地方的代码根本不需要改动。

假设你开始用ArrayList alist = new ArrayList(), 这下你有的改了，特别是如果你使用了ArrayList实现类特有的方法和属性。

这样的好处是为了代码的可维护性，可复用性，可扩展性以及灵活性，再者就是这符合了里氏代换原则和开闭原则。

————————————————

版权声明：本文为CSDN博主「longHARDEN」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/qq\_42002500/article/details/111088801

#### .拆箱和装箱boxing unboxing

* 装箱:将值类型装换为引用类型
* 拆箱:将引用类型转换为值类型

代码中应该避免装箱和拆箱

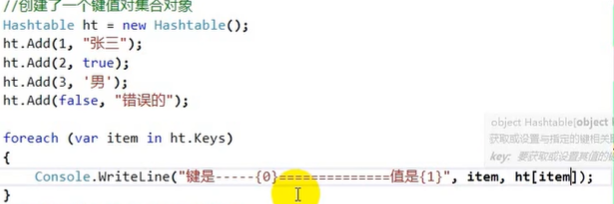
* 看两种类型是否发生了装箱或者拆箱,要看,这两种类型是否存在继承关系.

### Hachtable键值对集合





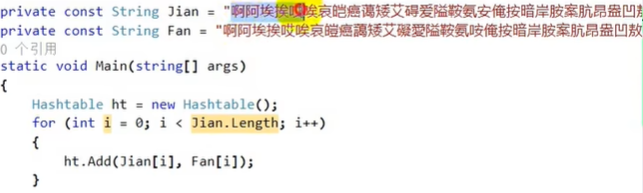
* 用foreach遍历Hachtable集合





Var是猜测里面的东西是什么类型,弱语言

* 繁简转换



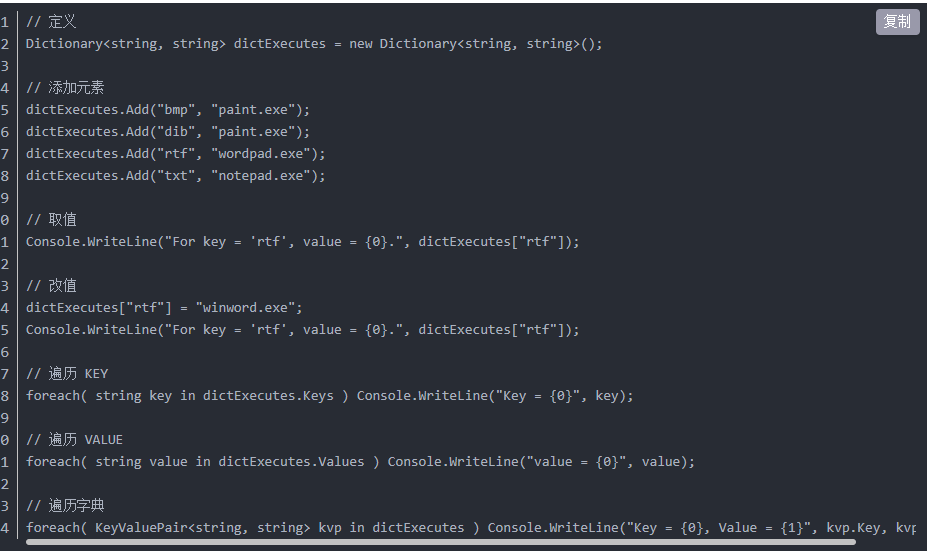


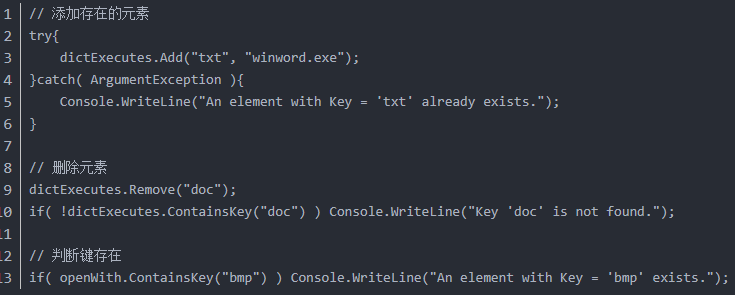
#### 字典集合dictionary

* 必须包含名空间 System.Collection.Generic
* Dictionary里面的每一个元素都是一个键值对（由二个元素组成：键和值）
* 键必须是唯一的,而值不需要唯一的
* 键和值都可以是任何类型(比如：string, int, 自定义类型等等)

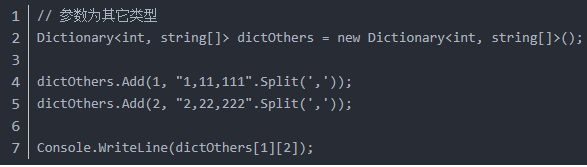
可以简单将 Dictionary 理解为 键值对 数据的集合

**常用方法:**

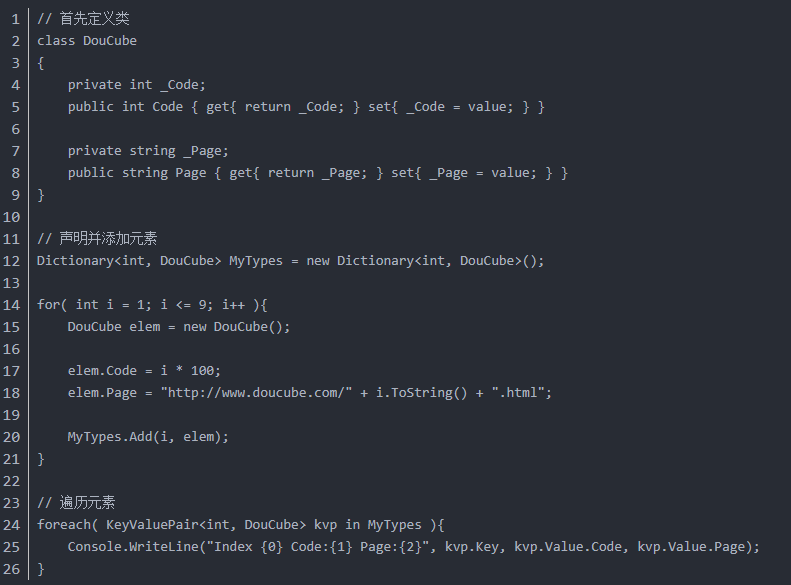




**参数为其他类型**



参数为自定义类型



————————————————

版权声明：本文为CSDN博主「外来物种」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/dmlk31/article/details/111206272

# 二 方法

## C# Methods

方法是给代码运行的地方,只有被调用时才运行

使用方法是因为: 想重复利用里面的功能

### 创建方法:名字加括号

小规则:类和方法名称首字母大写

class Program

{

static void MyMethod()

{

// code to be executed

}

}

MyMethod():函数名

static: 说明方法是属于类Program的,而不是Program的一个对象

void: 说明方法不返回值

## C# Method Parameters 方法中的参变量

方法可以定义很多个变量,只需要用逗号隔开

### Parameters and Arguments

static void MyMethod(string fname) //方法,定义了名叫fname的参变量

{

Console.WriteLine(fname + " Refsnes");

}

static void Main(string[] args) //调用方法的方法

{

MyMethod("Liam");

MyMethod("Jenny");

MyMethod("Anja");

}

// Liam Refsnes

// Jenny Refsnes

// Anja Refsnes

* When a **parameter** is passed to the method, it is called an **argument**. So, from the example above: is a **parameter**, while , and are **arguments**.fnameLiamJennyAnja

当一个参数传递给方法时,它叫做一个argument.

* Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

请注意，在使用多个参数时，方法调用必须具有与参数相同的参数数，并且必须以相同的顺序传递这些参数。

### Default Parameter Value 默认参数值

可以用一个默认值定义参数 用等号=

当函数被调用但是没有给参数赋值时,他就会用默认值

**不管是形参还是实参,声明时都会创建空间**

### Return Values 返回值

不用void

为返回值定义数据类型 例如 int double 来代替void ,用return 放在方法体中

static int MyMethod(int x)

{

return 5 + x;

}

static void Main(string[] args)

{

Console.WriteLine(MyMethod(3));

}

// Outputs 8 (5 + 3)

### Named Arguments 命名参数

除了=

也可以用 :

static void MyMethod(string child1, string child2, string child3)

{

Console.WriteLine("The youngest child is: " + child3);

}

static void Main(string[] args)

{

MyMethod(child3: "John", child1: "Liam", child2: "Liam");

}

// The youngest child is: John

### Method Overloading 方法重载

* 使用方法重载.不同的方法可以使用相同的参变量名字
* Instead of defining two methods that should do the same thing, it is better to overload one.

In the example below, we overload the method to work for both and :PlusMethodintdouble

static int PlusMethod(int x, int y)

{

return x + y;

}

static double PlusMethod(double x, double y)

{

return x + y;

}

static void Main(string[] args)

{

int myNum1 = PlusMethod(8, 5);

double myNum2 = PlusMethod(4.3, 6.26);

Console.WriteLine("Int: " + myNum1);

Console.WriteLine("Double: " + myNum2);

}

### 方法的递归

1. 方法自己调用自己.
2. 必须设置停止递归的方式return.
3. 返回的时候要一层层出去

* 静态字段 public static int i=0;

在所有函数之外,类之内声明.所有的函数都能访问并改变该静态字段.适用于方法递归.

# 三 类

## C# OOP Object-Oriented Programming

Object-Oriented Programming以对象为导向的编程方式

* OOP is faster and easier to execute 快 简单编程
* OOP provides a clear structure for the programs 清晰的结构
* OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug \\”不要重复自己”
* OOP makes it possible to create full reusable applications with less code and shorter development time 缩短开发时间 代码更短

**Tip:** The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it. 不要重复自己(DRY)的宗旨是:减少代码的重加载,把代码放在一个位置上然后重复利用它而不是重复这段代码

## C# Classes and Objects 类和对象

对象是根据类创建出来的

### Classes and Objects

举例:

类class: 车

对象object: 桑塔纳,大众,丰田

属性attributes: 颜色,高度

方法methods: 开车drive, 刹车brake

#### 创建类 创建类需要用到关键词 class

Create a class named "" with a variable :Carcolor

创建一个叫Car的类,包含一个变量color

class Car

{

string color = "red";

}

当一个变量在类中被定义,它叫做一个**“域““field“(字段)**

**通常类名和文件名是一样的,这让代码更易读**

**类不占内存,对象占内存**

**类class和结构struct:结构也有属性和方法,但是结构不具备类的任何一个特征(继承,多态…)**

#### 创建对象

对象在类里被创建

„类名 对象名 = new 类名“

class Car

{

string color = "red";

static void Main(string[] args)

{

Car **myObj** = new Car();

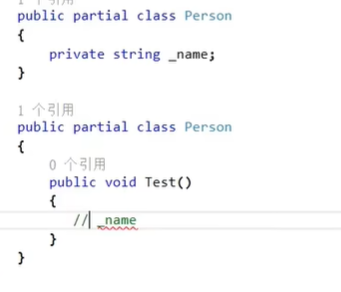
Console.WriteLine(myObj.color);

}

}

### 部分类partial

把一个类分成多个部分写



### 密封类sealed

特征:不让继承,可以继承别人

### Multiple Objects多个对象

创建多个对象并调用他们的属性

调用属性: 对象名.属性 (myObj1.color)

class Car

{

string color = "red";

static void Main(string[] args)

{

Car myObj1 = new Car();

Car myObj2 = new Car();

Console.WriteLine(myObj1.color);

Console.WriteLine(myObj2.color);

}

}

使用多个对象

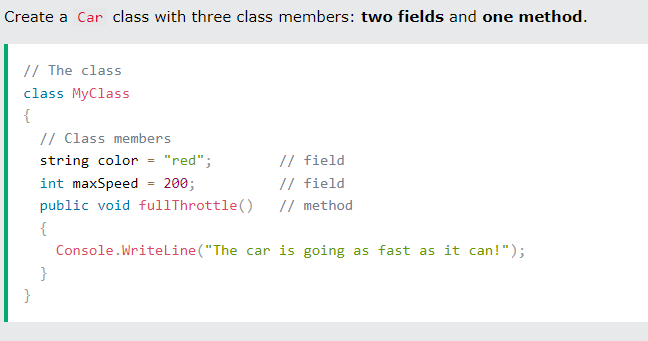
我们可以定义一个类但是在别的类中使用它 , 这通常是更好的管理类的方法(一个类拥有所有的域和方法,另一个类用来实现它)

使用 **public**关键词来让变量/域开放 而不是**static**



## C# Class Members 类的成员

Class Members 类成员通常理解为: 域 和 方法

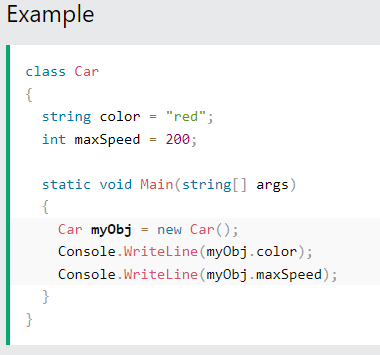


域 1 (属性1)

域2 (属性2)

类方法

### 域 Field

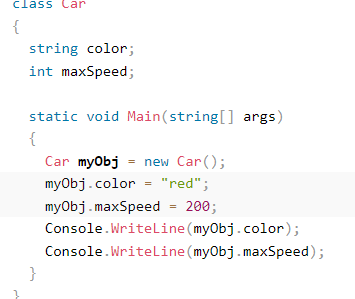


调用域 使用(**.**)

创建域

域的值也可以不赋值,在后边创建对象的时候给对象的属性(域的映射)单独赋值

麻烦 不用

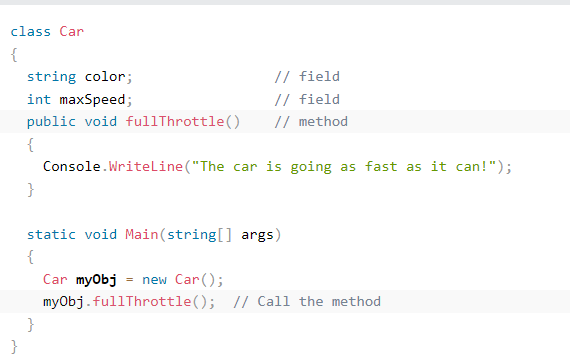


### Object Methods对象方法 (对象调用的方法)

Methods normally belongs to a class, and they define how an object of a class behaves. 方法通常属于类, 它注明了对象在类中的功能(车的启动刹车功能)

类中的方法定义:方法一定要标注上public

记住: 调用时有括号()和分号 ;



调用一下~

类中方法定义

没有分号的哈

### Use Multiple Classes在多个类中调用

看看就行,还是用constructor好一些



一个一个定义麻烦死啦!

## C# Constructors 构造函数

* A constructor is a **special method** that is used to initialize objects.

The advantage of a constructor, is that it is called when an object of a class is created. It can be used to set initial values for fields:

构造函数是一种初始化对象的**特殊方法;**

构造函数的优势是:当对象被创建的时候,能被调用并且给对象的值进行初始化(不用手动定义啦就是说!)

* Note that the constructor name must **match the class name**, and it cannot have a **return type** (like or ). void int

Also note that the constructor is called when the object is created.

All classes have constructors by default: if you do not create a class constructor yourself, C# creates one for you. However, then you are not able to set initial values for fields.

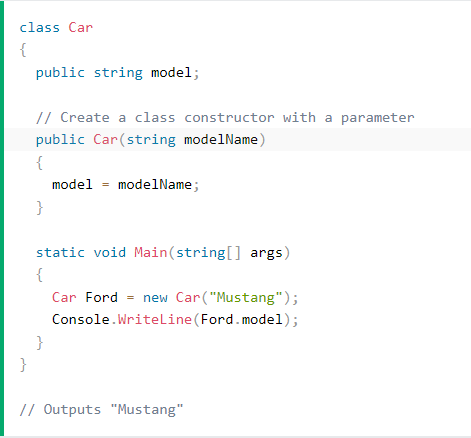
**Constructors save time!** Take a look at the last example on this page to really understand why.

构造函数名必须和类名一样(不然怎么调用呐!)而且不能有返回值

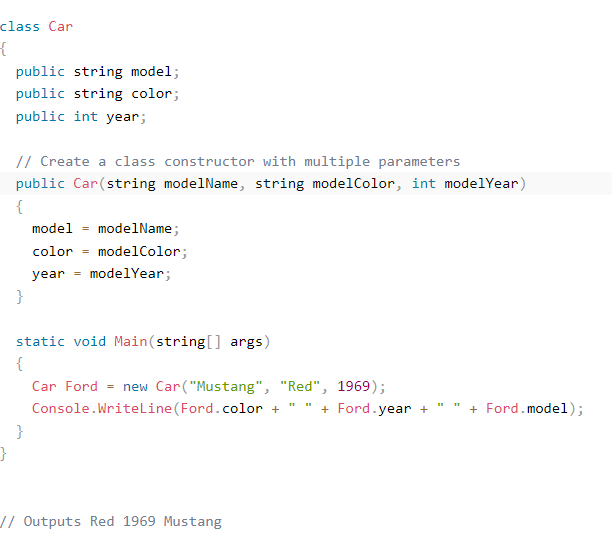
记住: 每创建一个对象 , 构造函数就被调用一次

每个类都有初始构造函数,如果你不创建,C#就自动给你创建.(听着还挺好的吧,但是!)你就不能给域手动初始化了

**构造函数很省时间!**看例子:



直接给名字



创建的时候直接赋值~爽歪歪

属性都放进去~

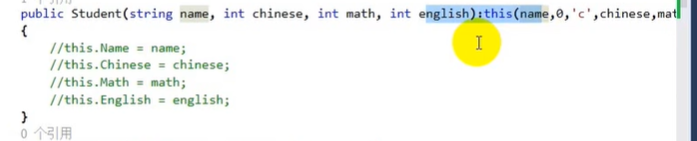
### this关键字

[C# |这个关键词 - 极客极客 (geeksforgeeks.org)](https://www.geeksforgeeks.org/c-sharp-this-keyword/)

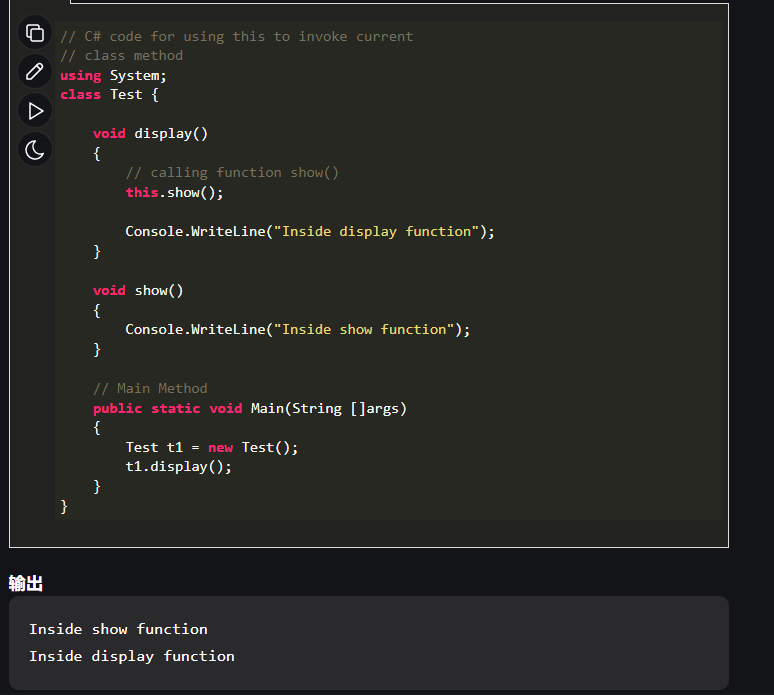
1.调用时,代表当前类中的对象

2.在类当中显示的调用本类的构造函数. :this()

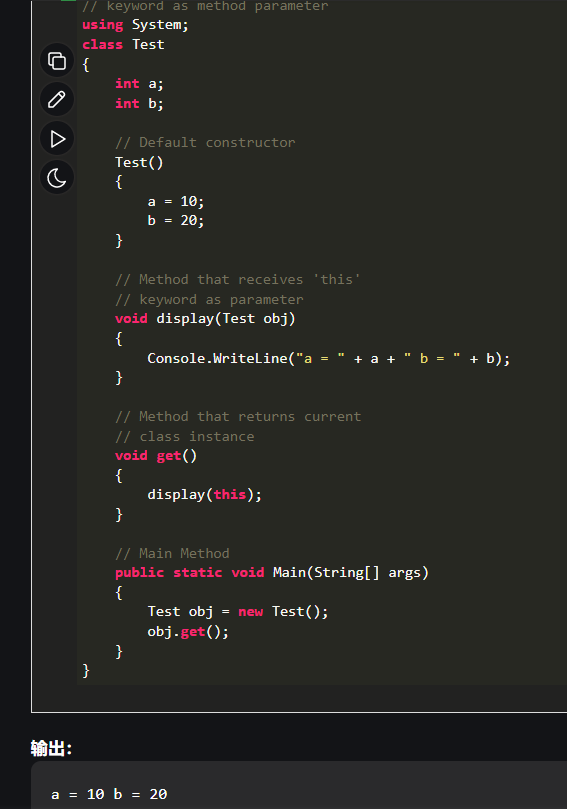
调用已经有的构造函数,省去再写一遍



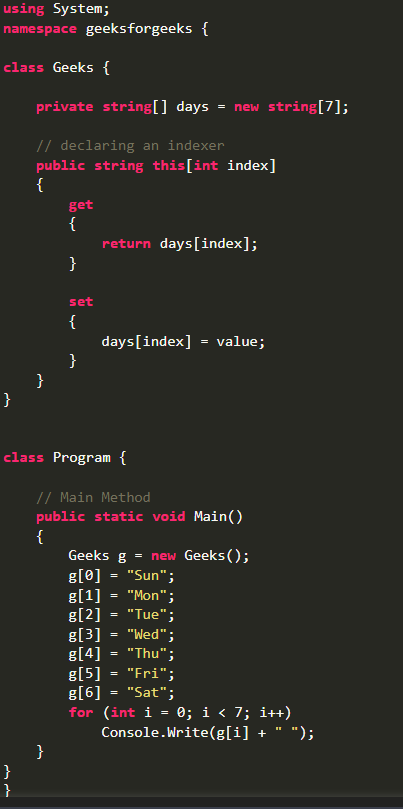
1. 调用当前类中的方法



1. 使用this作为函数的方法参数(method parameter)



1. 作为 声明索引器



### 析构函数

一般来说,程序结束时才会释放资源

使用析构函数可以立即释放

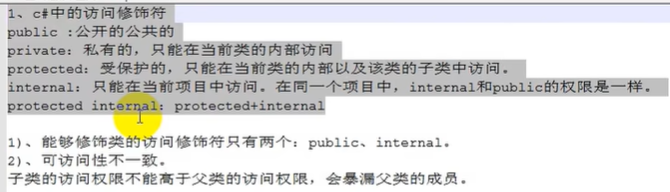
~Student()

{

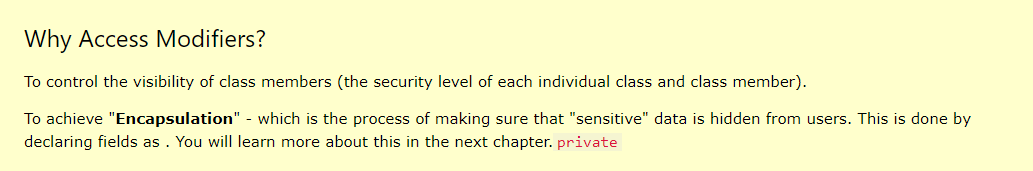
}

## C# Access Modifiers 访问权限修饰符

|  |  |
| --- | --- |
| **Modifier** | **Description** |
| public | The code is accessible for all classes 所有类可调用 |
| private | The code is only accessible within the same class 同类可调用 |
| protected | The code is accessible within the same class, or in a class that  is inherited from that class.同类可调用,或者继承类可调用. |
| internal | The code is only accessible within its own assembly,  but not from another assembly. 只能在当前项目中访问  **在同一个项目中, internal和public的权限是一样的,internal权限大于protected**  **在不同项目中,protected权限大于internal,因为只要外部项目中的类继承了现在项目的一个protected类,这个protected类中的东西就能被外部类调用.而internal的所有东西都不能用了.** |
| protected internal | Protected + internal |
| private protected | Private protected |



为什么使用权限修饰符--------------为了实现**封装**~~数据保护



## C# Properties (Get and Set) 属性 财产? (外部如何得到内容和设置成隐私)

* The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:
* declare fields/variables as private
* provide public get and set methods, through **properties**, to access and update the value of a private field
* 属性可以看做变量和方法的结合,并且内含两种方法: get set

定义属性代码:

class Person

{

private string name; // field

public string Name // property **没有括号的哈,代表了域**

{

get { return name; } // get method

set { name = value; } // set method

}

}

例:



注 : 最好把property和变量用同样的名字,用大小写区分Name name

这个方法的返回值是变量的值. get name

The method assigns a to the variable. The keyword represents the value we assign to the property. set value name value

Why Encapsulation?

* Better control of class members (reduce the possibility of yourself (or others) to mess up the code)更好的控制类中的成员(减少自己把代码弄混的可能)
* Fields can be made **read-only** (if you only use the get method), or **write-only** (if you only use the set method)域可以被设置成**只读**(如果只用get方法)或只能**编辑**(如果只用set方法)
* Flexible: the programmer can change one part of the code without affecting other parts

灵活:编程者可以改变代码的一部分,而不必影响其他的部分

* Increased security of data 增强代码安全性

### 属性

属性的作用就是保护字段`对字段的赋值和取值进行限定

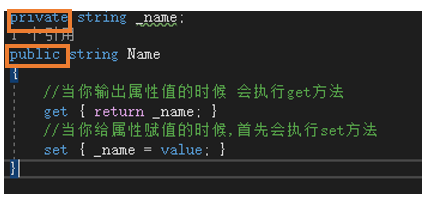
字段非常重要,外界应该只能访问到属性

属性的本质就是两个方法,一个叫get()一个叫set()

既有get()又有set()的属性叫可读可写属性

只有get()没有set()的属性叫只读属性

把字段保护起来,用private



## C# Inheritance 继承

1. 单根性
2. 传递性

In C#, it is possible to inherit fields and methods from one class to another. We group the "inheritance concept" into two categories:

* **Derived Class** (child) - the class that inherits from another class
* **Base Class** (parent) - the class being inherited from

To inherit from a class, use the : symbol.

In the example below, the Car class (child) inherits the fields and methods from the Vehicle class (parent):

class Vehicle // base class (parent)

{

public string brand = "Ford"; // Vehicle field

public void honk() // Vehicle method

{

Console.WriteLine("Tuut, tuut!");

}

}

class Car : Vehicle // derived class (child)

{

public string modelName = "Mustang"; // Car field

}

class Program

{

static void Main(string[] args)

{

// Create a myCar object

Car myCar = new Car();

// Call the honk() method (From the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand field (from the Vehicle class) and the value of the modelName from the Car class

Console.WriteLine(myCar.brand + " " + myCar.modelName);

}

}

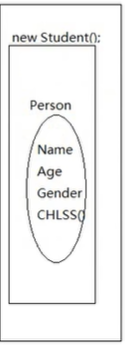


可以看到,子类继承了母类的公开属性和方法,并且可以调用

子类并没有继承父类的构造函数,但是子类会默认调用父类无参构造函数

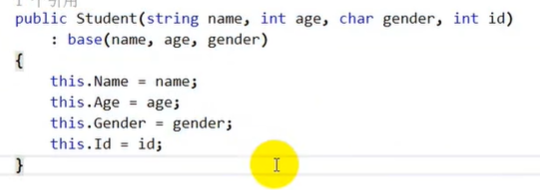
因为:当我们创建子类对象时,首先会创建一个父类对象

但是子类并没有继承父类的私有字段



#### :base()

如果想调用有参的父类构造函数,用:base()

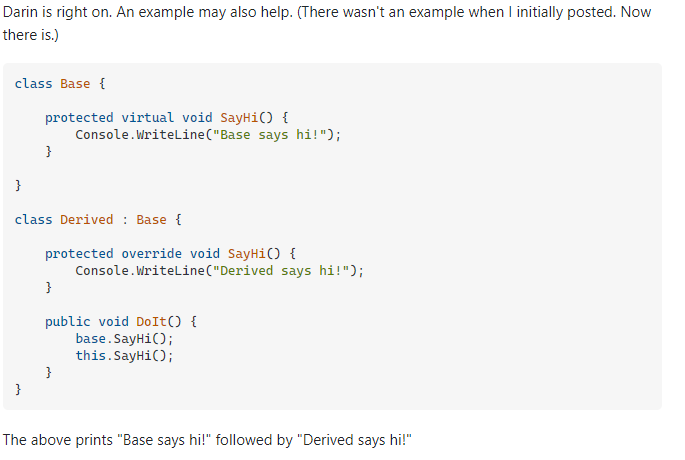


只需要给多出来的属性赋值就行

**this 和base的区别**

base - 用于从派生类中访问基类的成员,

this - 引用类的当前实例并继承



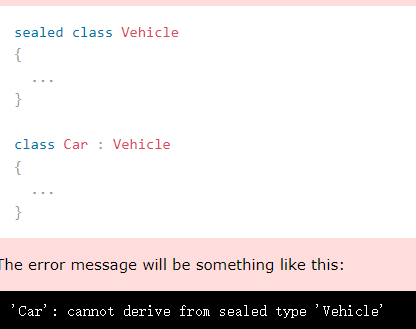
* **object类是所有类的基类**
* Why And When To Use "Inheritance"?

- It is useful for code reusability: reuse fields and methods of an existing class when you create a new class.

**Tip:** Also take a look at the next chapter, [Polymorphism](https://www.w3schools.com/cs/cs_polymorphism.php)(多态), which uses inherited methods to perform different tasks.

### The sealed Keyword

If you don't want other classes to inherit from a class, use the sealed keyword:



### New隐藏父类成员

1. 创建对象
2. 隐藏从父类那里继承过来的同名成员

后果:调用不到父类成员

### 静态和非静态的区别

1. 在非静态类中.既可以有实例成员, 也可以有静态成员
2. 在调用实例成员的时候,需要使用 对象名.实例成员;

在调用静态成员的时候,需要使用 类名.静态成员;

总结: 静态成员必须使用类名调用,而实例成员使用对象名调用.

静态函数中,只能访问静态成员,不允许访问实例成员.

实例函数中,既可以使用静态成员也可以使用实例成员.

静态类中只允许有静态成员,不允许有实例成员.

使用:

1. 如果你想要你的类当做一个‘工具类“去使用,这个时候可以考录将类写成静态类(不需要创建对象)
2. 静态类在整个项目中资源共享.(静态类占内存,普通类不占内存,对象占内存)
3. 用户登录QQ.

释放资源:GC Garbage Collection垃圾回收器

1. 只有在程序全部结束之后,静态类才会释放资源.(空间 农场 微博…全部关掉)
2. 静态类越少越好

### 值类型和引用类型 (数据类型)

区别:

1. 值类型和引用类型在内存上存储的地方不一样.
2. 在值传递和引用传递的时候,传递的方式不一样

值类型我们称之为值传递,引用类型我们称之为引用传递.

我们学的值类型: int, double, bool, char, decimal, struct, enum

引用类型:string, 自定义类, 数组,接口

存储:

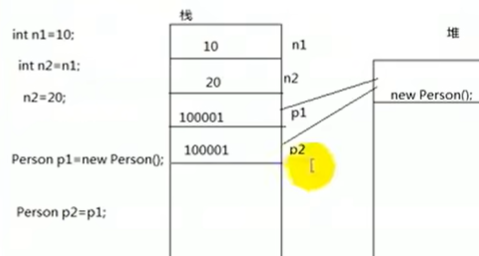
值类型的值是存储在内存的栈当中.

引用类型的值是存储在内存的堆当中,它的地址储存在栈当中,两者通过地址连接.

复制:

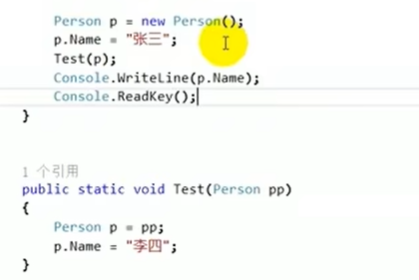
值类型复制的时候传递的是值本身

引用类型在复制的时候,传递的是对这个对象的引用.



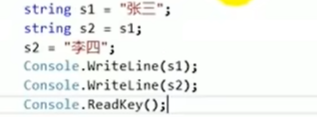
P1和P2指向同一块空间, 当一个改变, 另一个的值也会改变.

举例1:



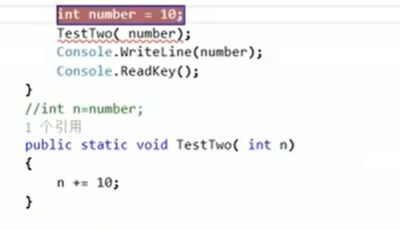
结果为“李四“

举例2:



结果为“张三,李四

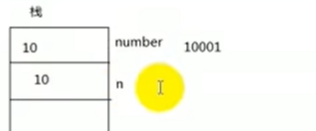
举例3:



结果为n=10

原因:

TestTwo作用的是形参n,它也有一个空间存储地址,在函数内部改变的是n的值,而不是number的值. 要改变,用ref.



**字符串具有不可变性**

### 里式转换Liskov 替换原则 强转

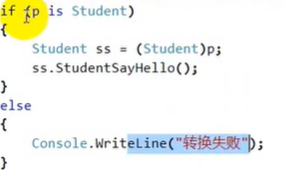
1. 子类可以赋值给父类:如果有一个地方需要一个父类作为参数,我们可以给一个子类代替



1. 如果父类装的是子类对象,那么可以将这个父类强转为子类对象



1. is:表示类型转换,如果能够转换成功,则返回TRUE,不能转换返回FALSE



as: 表示类型转换,如果能够转换成功,则返回转换成功的对象,



### 序列化和反序列化

序列化:将对象转换为二进制

反序列化:将二进制转换为对象



序列化的方法

用作序列化的类

创建一个文件,把序列化之后的对象存储进去



反序列化的方法,加个de

强转为person类

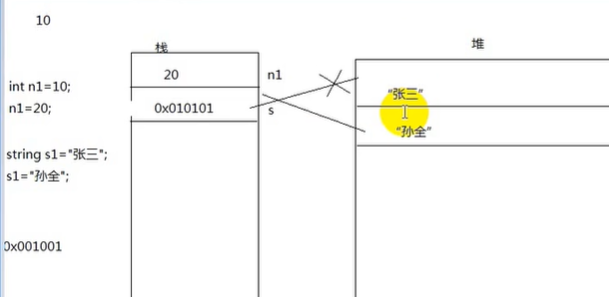
用反作序列化的类,是一样的

### String字符串

#### 字符串的不可变性:

当给字符串重新赋值后,老值并没有被销毁,而是在堆中开辟新空间.在栈中存储新的地址.

在程序结束后,GC扫描整个内存,如果发现有的空间没有被指向,则立即把它销毁.

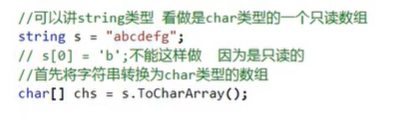


#### 字符串的方法

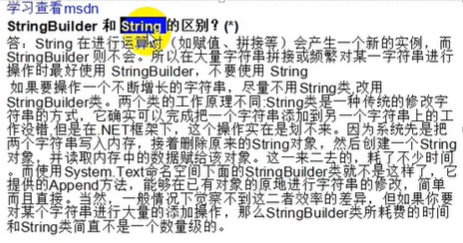
我们可以将字符串看做是char类型的一个只读数组.



1. 用char[] ToCharArray(),可以把string改成char数组.



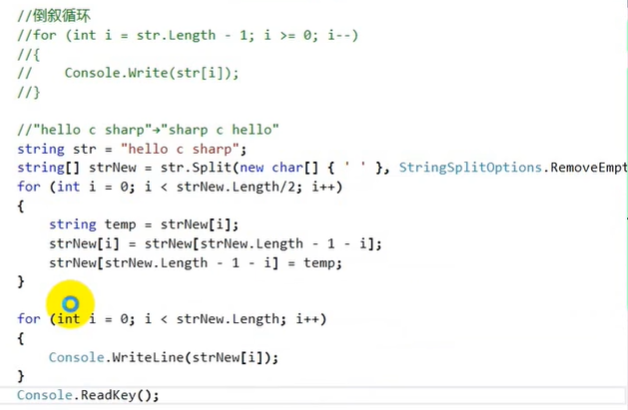
1. StringBuilder()当一个字符串反复大量的被修改时用

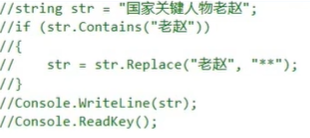
1. string ToUpper().ToLower():

将字符串转换成大写形式.ToUpper()和小写形式.ToLower()

1. A.Equals(B, StringComparison,OrdinalIgnoreCase,)字符串值是否相等,忽略大小写
2. Split()分割字符串



1. bool Contains(string value) s.Replace(string value, string value) 找到字符串并替换成另一个



1. s.SubString(int n)从n位开始截取字符串,包括当前位置
2. bool StartsWith(string value)判断字符串是否以子串value开始;

bool EndWith(string value) 判断字符串是否以子串value结束;

bool Contains(string value)判断字符串含不含字串

1. int IndexOf(string value,int n)判断字符串第一次出现的位置,从第n个位置开始找,包括当前位置,如果没有返回-1



1. int LastIndexOf(string value)判断字符串最后一次出现的位置, 如果没有返回-1
2. string Trim()移除字符串前后的空格

string TrimStart()移除所有的前面的空格

string TrimEnd()移除所有的结尾的空格

1. string.IsNullOrEmpty() 判断是否为空字符串
2. string.John(string separator,string)把想要的分隔符放在字符串数组里,返回一个新字符串.

## C# 多态和覆盖Polymorphism and Overriding Methods

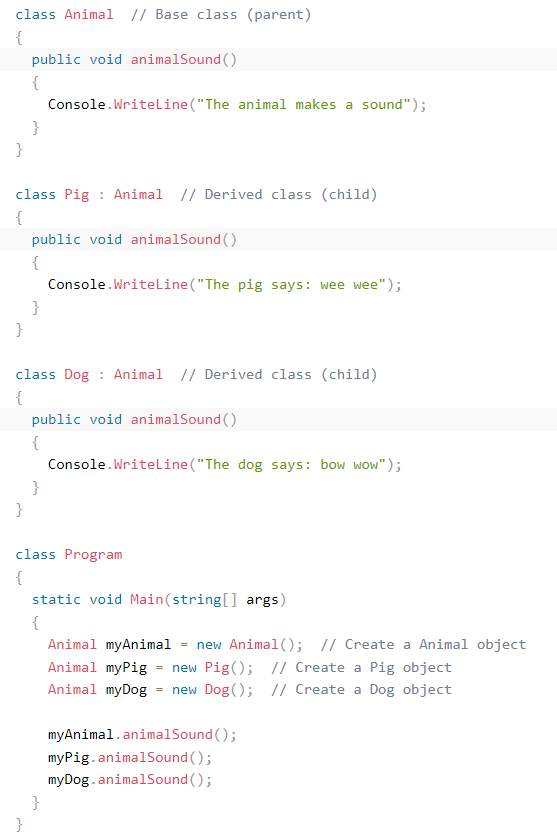
继承:从母类中继承域(属性)和方法

多态:每个子类从母类中继承方法,但拥有不同的功能

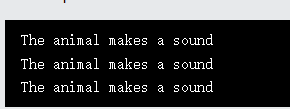
实现多态:关键字: virtual override 虚拟 覆盖

[区别]

* 从几个类中能抽象出一个父类,且父类不知道如何实现→抽象类
* ,且父类也有自己的实现→虚方法
* 不能抽象出父类,但子类有共同的一个行为→接口

错误示范:无多态



### 多态之虚方法(from 老赵)

概念:让一个对象能够表现出多种的状态

实现多态的3种方法:1.虚方法2.抽象类3.接口

1.虚方法: 父类的方法标记为虚方法 , 使用virtual ; 子类方法前加override(调用的依然是父类方法,但是被子类重写了.使用哪个子类方法取决于当前对象属于哪个子类)

2.抽象类: 当父类中的方法不知道如何去实现的时候,可以把父类写成抽象类,方法为抽象方法.

* 抽象成员必须标记为abstract ,并且不能有任何实现.
* 抽象成员必须在抽象类中
* 抽象类不能被实例化.
* 继承抽象类的子类必须重写父类的所有抽象成员(抽象方法,抽象参数….)

除非子类也是抽象类,可以不重写

* 抽象类成员的访问修饰符不能是private
* 抽象类中可以包含实例成员.

并且抽象类的实例成员可以不被子类实现

* 如果父类抽象方法有参数和返回值,子类在实现的时候也要有相同的返回值和参数

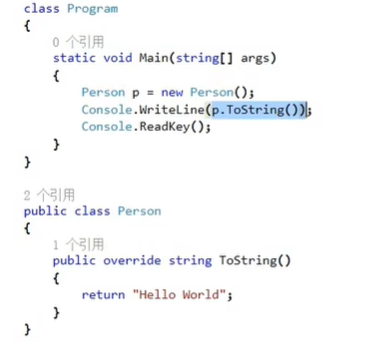
**[区别和使用情况]** 虚方法里可以有函数体(父类函数有意义),抽象类不允许有函数体.

如果父类的方法有默认的实现,并且父类需要被实例化,这时可以考虑将父类定义为一个普通类,用虚方法来实现多态

如果父类中的方法没有默认实现,父类也不需要被实例化,则可以考虑将该类定义为抽象类.

### 重写父类的ToString

ToString()是Object类的虚方法,所有类都可以调用.当然也可以改写



## C# Abstraction 抽象 抽象类

Abstract Classes and Methods抽象类和抽象方法

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.隐藏细节 只展示最重要的信息  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/cs/cs_interface.php) (which you will learn more about in the next chapter).抽象可以通过构造抽象类或者**接口**实现

使用 **:** 继承抽象类, 使用 **override** 覆盖(重载)抽象方法

The keyword is used for classes and methods: abstract

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).抽象类不能用来创建对象(想访问必须通过其他类的继承)
* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).(抽象方法只能在抽象类里用,无函数体.主体由派生类(继承)提供)

An abstract class can have both abstract and regular methods:

抽象类可以同时拥有抽象方法和常规方法:



覆盖原抽象方法

并构建函数体

派生类Pig

**Why And When To Use Abstract Classes and Methods?**

To achieve security - hide certain details and only show the important details of an object.

**Note:** Abstraction can also be achieved with [Interfaces](https://www.w3schools.com/cs/cs_interface.php), which you will learn more about in the next chapter.

## C# Interface 接口

### Interface

* 接口是一个“完全抽象类“,只能包括抽象方法和属性(没有体),没有属性: interface
* 在接口命名是最好加上“I“来提醒自己,这是一个接口,不是类
* 通常来说,接口的默认成员是 abstract public (所以下面例子没有写这俩)
* 接口的实现只能通过继承
* 使用 : 来继承, 但是不使用override(试过了,会出错)

// interface

interface IAnimal

{

void animalSound(); // interface method (does not have a body)

void run(); // interface method (does not have a body)

}

Notes on Interfaces:

Like **abstract classes**, interfaces **cannot** be used to create objects (in the example above, it is not possible to create an "IAnimal" object in the Program class)和抽象类一样,接口不能用来创建对象(因为没有方法体,无法实现)

Interface methods do not have a body - the body is provided by the "implement" class接口中的方法没有体,体在”实现”类中

On implementation of an interface, you must override all of its methods要实现接口,必须重载所有的方法

Interfaces can contain properties and methods, but not fields/variables接口可含属性和方法,但是不含域(参数/自动属性),他们本质上都是方法.见下面的自动属性和普通属性.

Interface members are by default abstract and public 接口所有成员默认abstract and public

An interface cannot contain a constructor (as it cannot be used to create objects) 接口中不可以有构造函数

Why And When To Use Interfaces?

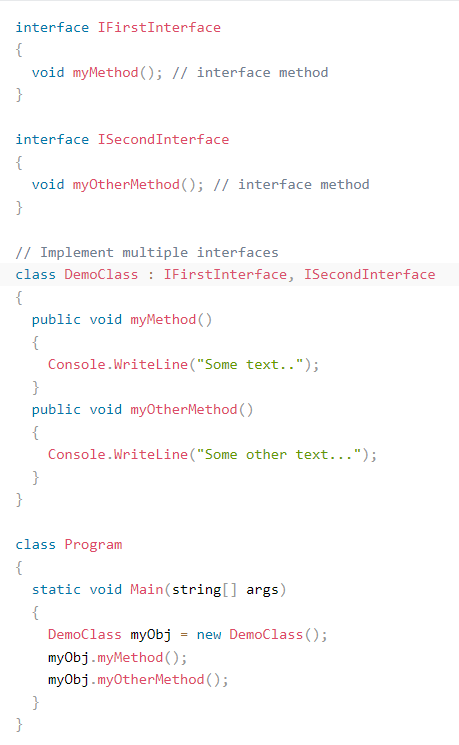
1) To achieve security - hide certain details and only show the important details of an object (interface).

2) C# does not support "multiple inheritance" (a class can only inherit from one base class). However, it can be achieved with interfaces, because the class can **implement** multiple interfaces. **Note:** To implement multiple interfaces, separate them with a comma (see example below).C#中没有多继承(一个子类只能继承一个母类)但是可以继承很多个接口,用**: .**

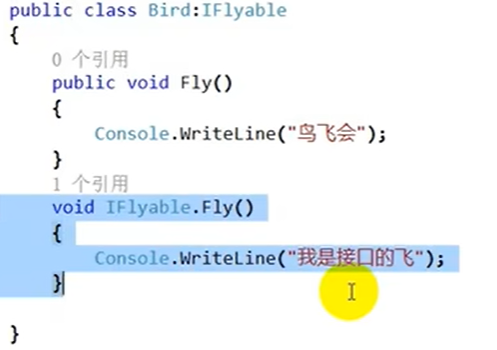
使用接口和构建抽象类的区别??

### Multiple Interfaces多元接口

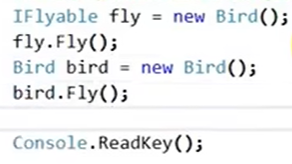
继承多个接口只需要用很多个**:** 把他们隔开



### 显式接口就是为了解决方法的重名问题



重名的话,在子类中显式重写接口



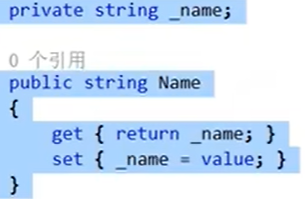
调用自己类中重名的方法

调用接口的方法

### 自动属性和普通属性

[共同点]C#都会自动添加字段和方法体在后台

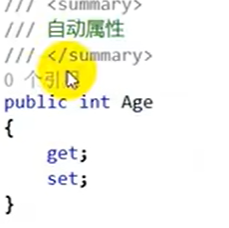
普通属性:有字段,有方法体



自动属性:无字段,无方法体.

优点:方便

缺点:不能构建方法体,只能通过构造函数



## C# Enum 枚举

* 为什么以及何时使用枚举？

当您知道值不会更改时，请使用枚举，例如月日，日，颜色，一副牌等。

含义:特别列出

* Enum 是一个特殊的“类”，表示一组**常量**（不可更改/只读变量）。enum

### 要创建 ，请使用关键字（而不是类或接口），并用逗号分隔枚举项：enum enum

enum Level

{

Low,

Medium,

High

}

可以使用**点**语法访问项目：enum

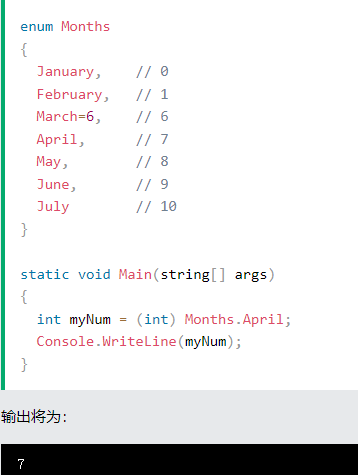
Level myVar = Level.Medium;

Console.WriteLine(myVar);

### 枚举值

默认情况下，枚举的第一项的值为 0。第二个的值为 1，依此类推。

若要从项中获取整数值，必须将项[显式转换为](https://www.w3schools.com/cs/cs_type_casting.php) ：int

自己分配枚举值,接下来的项目像相应的更新数字

默认从0开始

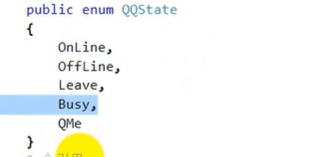
#### 常用 switch case



#### 枚举和遍历有什么区别???

### 枚举类型和int类型转化

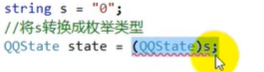
所有的类型都能转换成string类型

### 枚举类型和string类型转化



### 字符串转换成枚举类型

错误: 

原因:string和enum不兼容,不能直接转.

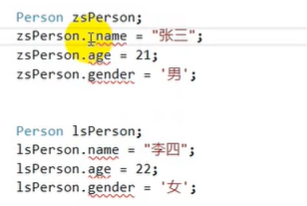
正确的: 

* 如果转化的字符串是数字,则就算枚举中没有,也不会抛异常:
* 如果转换的字符串是文本,如果枚举中没有,就抛异常

## C# 结构

定义:

赋值:



字段起名一定要加下划线

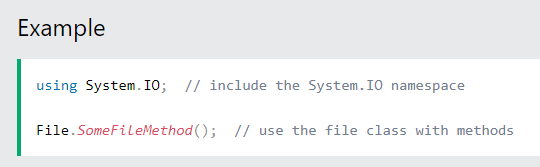
字段

外部访问权限

### 字段和变量的区别: 变量只能存储一个值,字段可以存储多个值

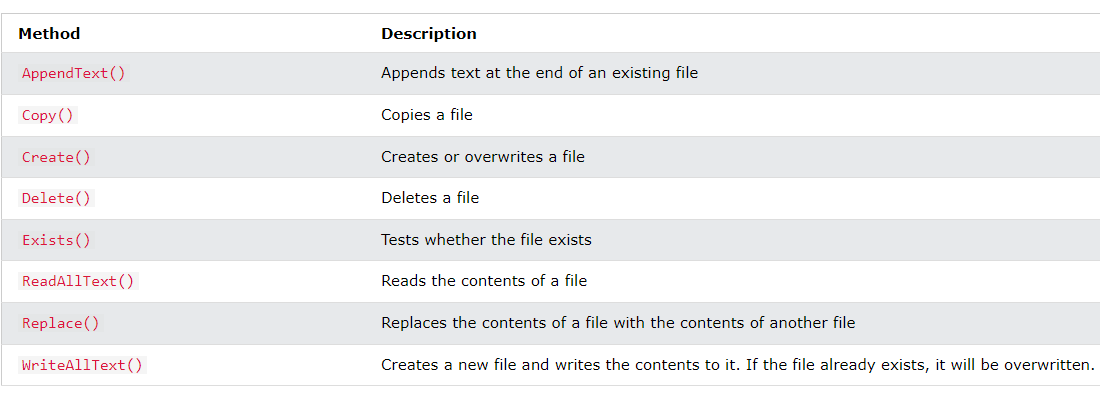
## C# Files 文件

File是命名空间里的类,允许我们直接使用File System.IO



每个类都有很多可以直接用的函数,举例: File 类

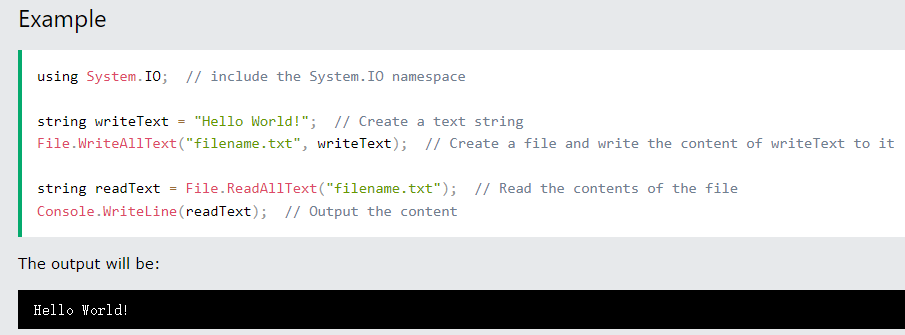
File中的所有方法可以去 [Microsoft .Net File Class Reference](https://docs.microsoft.com/en-us/dotnet/api/system.io.file?view=netframework-4.8).查



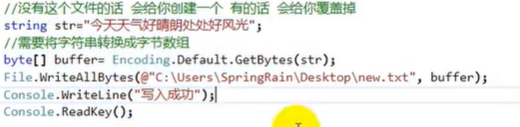
只能读小文件

删除文件

写一个文件并且读



字符串转换成字节数组,按字节写入



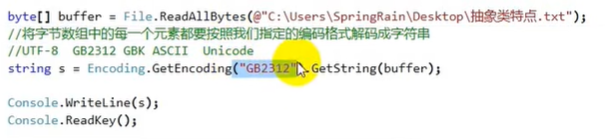
* 编码简介

编码:将字符串以怎么样的形式保存为二进制

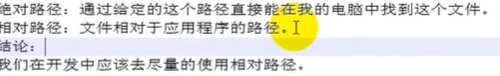
乱码:保存文件采用的编码和打开文件采用的编码不一样



解码

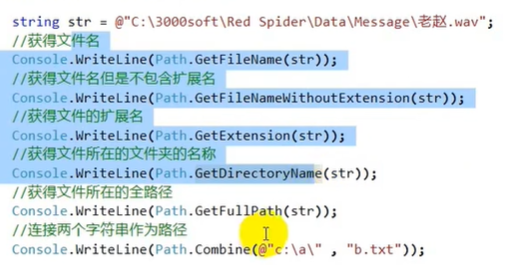


* 绝对路径和相对路径



* Path类

专门用来操作路径的



* **文件和文件流的区别**: File Stream是操作字节的,StreamReader和StreamWriter是用来操作字符的.如果想操作大文件, File Stream文件流或StreamReader 和StreamWriter,小文件可以用File.

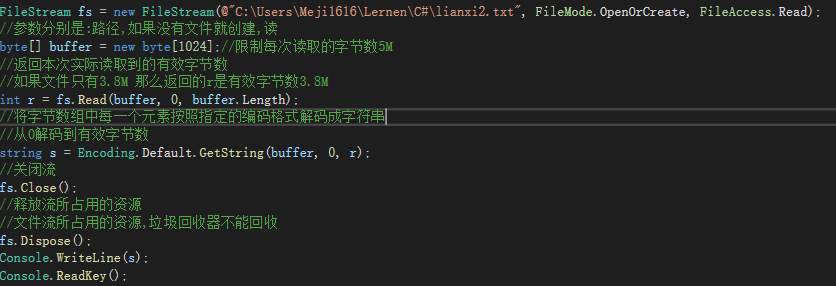
**FileStream可以操作所有的文件**

### File Stream文件流

使用FileStream读写文件

FileStream 继承的Stream类,是公开类,可以创建对象

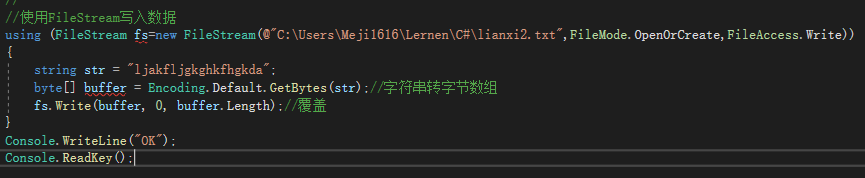
**读文件:**



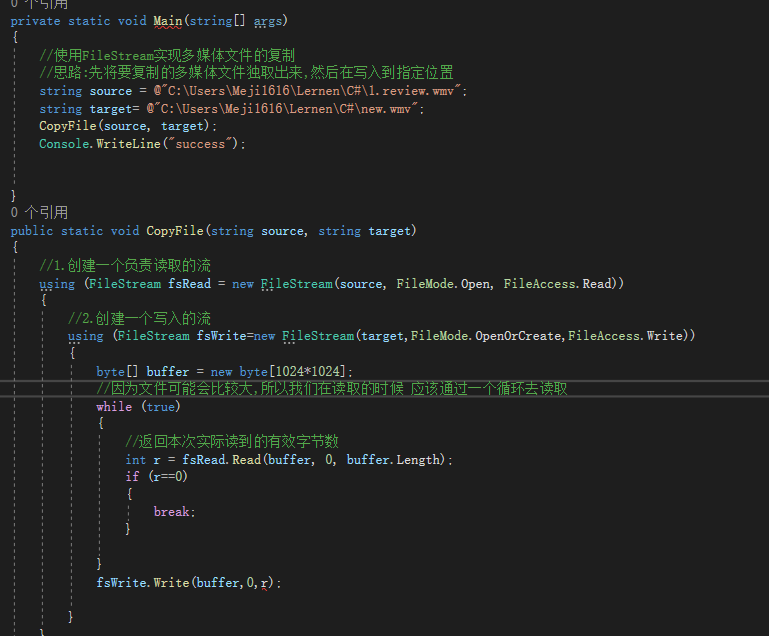
存储在buffer里

**写文件**

**使用using可以省去手动释放资源**

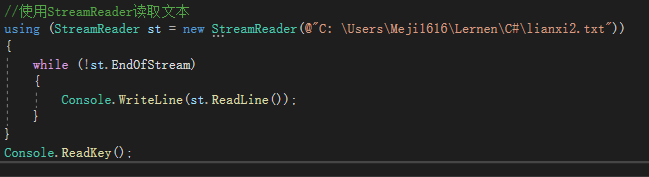
****

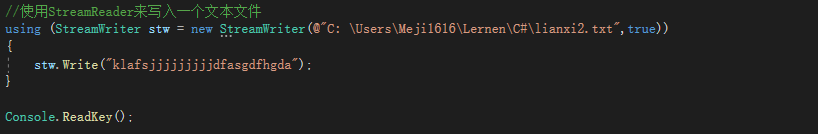
**使用FileStream实现多媒体文件的复制**

****

**应该在这里**

**StreamReader 和StreamWriter**

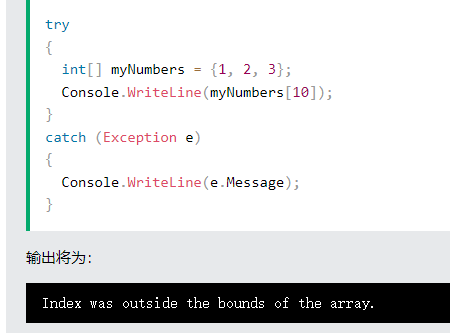
****

****

不覆盖

## C# Exceptions - Try..Catch 抓住异常

### 输出系统给的错误原因

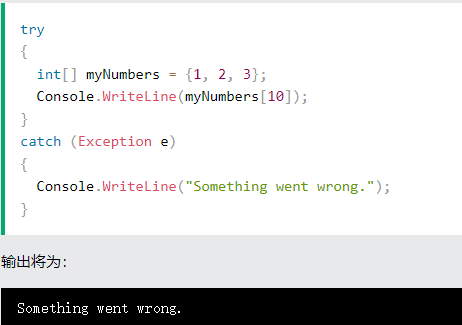


**Try中的代码执行到出错的那一行就不执行了.所以可以把出错后就不执行的剩下的代码都放到try里.\*(勤标注) 写完之后再全局考虑异常**

思路:找到应该放到try括号里的代码的最后一行,加上后括号},标注上这是什么的try,然后加上catch的代码,最后回到头部把try添加上.(这个思路也可以用在if else里和其他的可以break的代码)

**练习:输入年份月份判断有多少天**

### 输出自己定义的原因



#### 变量的作用域:

能够使用变量的范围

一般从声明它的那个括号开始到那个括号所对应的括号为结束

### 该语句允许您在 之后执行代码，而不管结果如何：finally try...catch



### 创建定义错误 throw

该语句允许您创建自定义错误。throw

该语句与**异常类**一起使用。C# 中有许多异常类可用：等：

throw ArithmeticException

FileNotFoundException

IndexOutOfRangeException

TimeOutException



# MD5 加密

# Winform application

## 练习题

### 如何让控制代码执不执行?

bool b= true //在整的代码前设bool类型true

try{

被检查的代码

}

catch{

报错代码

bool=false} //然后在会导致控制的代码不执行的代码后改变bool的值为false.

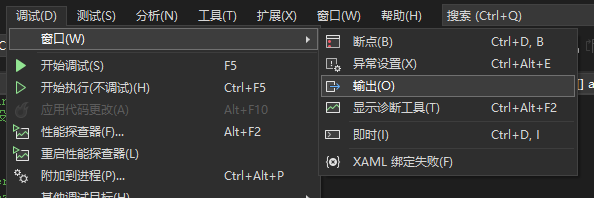
if (bool=ture) //然后再想控制的代码前加if条件,只有当bool=true才执行.

{要执行的代码}

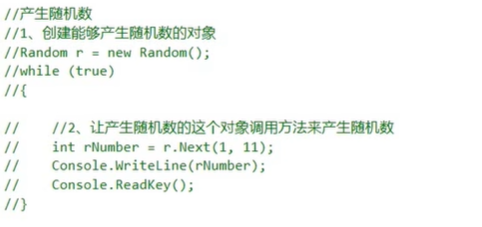
### 调试程序:

1. F11逐语句调试
2. F10逐过程调试
3. 断点调试

想看到参数的值:



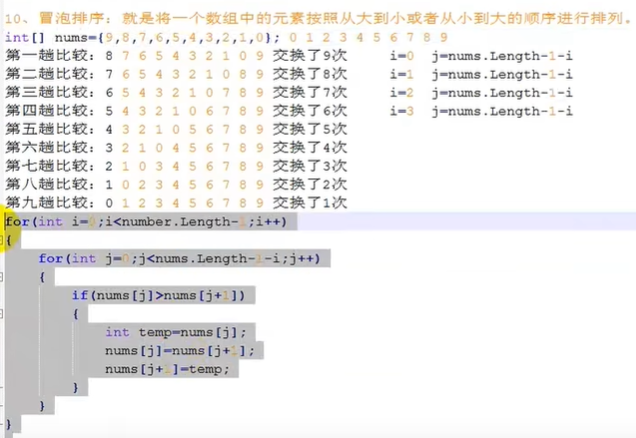
### 产生随机数 Random

取不到后边那个数

### 冒泡排序

升序排列:



## 设计模式:简单工厂设计模式

核心: 根据用户输入的创建对象赋值给父类.

[C-learning/简单工厂设计模式.md at main · NannF00/C-learning (github.com)](https://github.com/NannF00/C-learning/blob/main/%E7%BB%A7%E6%89%BF%E5%92%8C%E5%A4%9A%E6%80%81/%E7%AE%80%E5%8D%95%E5%B7%A5%E5%8E%82%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F.md)

## 飞行棋项目

### 1.画游戏头:每次更新都重新打印一次

### 2.初始化地图(加载地图所需要的资源)

把整数数组变成界面显示特殊字符串的过程,就是初始化地图

遍历数组中的每一个值(基本值为0),并设置特殊点的数字,当值为0则显示普通棋子,当值为特殊则显示对图案.



⌂

### 3.画地图