**面向对象程序设计预习指南**

最后一个NPE

# 第一章

**.NET Framework的构成**：

公共语言运行时（CLR）：是.NET Framework的基础，提供了程序代码跨平台运行的机制。

类库：可被.NET Framework语言调用，与CLR紧密集成的可重用类型集合。

**C#**:

概念：微软公司发布的一种简洁，类型安全的面向对象程序设计语言。

OOP:面向对象编程

特点：

1. 语言简洁。
2. 保留了c++强大的功能。
3. 快速应用开发功能。
4. 语言自由。
5. 强大web服务器控件。
6. 跨平台。
7. 与XML融合。

**VS2013:**

一款IDE。

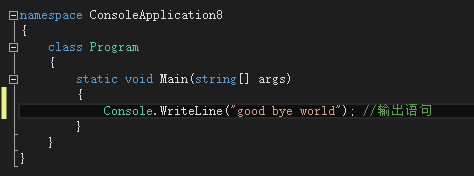
**C#程序基本结构**：

命名空间：组织各种成员的一种形式，类型java的包。

声明类：定义一个类成员。

Main方法：程序的入口点。

输入输出。

 注释。

命名空间

代码

声明类

Main方法

注释

**基本调试方法**：

F9设置断点。在调试的时候，程序将会在断点位置暂停。

F11逐语句调试。按一下这个键，程序执行流就会跳到下一条语句。程序会把一个函数调用也当做一条语句。

F10逐过程调试。按一下这个键，当执行到函数调用时，会跳入这个函数的内部。

F5进入调试状态。

# 第二章

**标识符**：

也就是名字。声明规则：由大小写字母、数字、下划线和@符合组成，区分大小写，不能以数字开头，也不能与C#的关键字相同。

**关键字**：

对编译器具有特殊意义的保留标识符。

**变量**：知道如何声明变量以及如何对变量赋值还有变量的作用域。

**常量**：知道如何声明常量。

常量只能在声明的时候初始化，并且一旦声明之后值就不能再改变，并且常量的值不能是变量。

**数据类型**：

值类型：

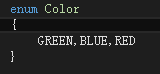
简单值类型。

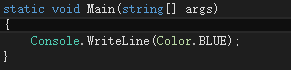
复合值类型：

结构类型。

枚举类型。

引用类型：类，接口，委托，数组。

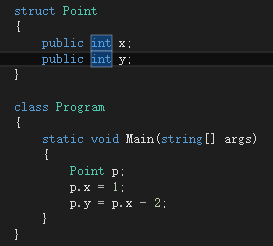
 **枚举**：

 可以看做是一组集合，如右侧的Color集合拥有GREEN/BLUE/RED三个集合元素。

枚举的使用

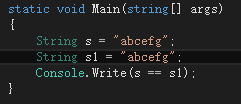
枚举的声明

**结构体**：

 是一种简单的值组合。

结构体的定义及其使用

**字符串**：

 知道字符串对象的常用方法：

字符串定义及使用

如IndexOf，SubString等等。

**类型转换**：

隐式转换：系统自己进行的，比如byte可以自动转换成int，short自动转换成int等等。

显式转换：C#中，显式转换一般是在值之前加上“（类型）”，或者是 “值 as 类型“ 。

拆箱装箱：

编译器可以自动进行值类型跟引用类型的自动转换。

C#中，如果一个函数调用需要一个Integer对象，那么可以向它传入一个int类型，编译器会自动将其从基本类型转换成引用类型这叫做装箱，反之叫做拆箱。

**运算符：**

基本运算符：

和C语言基本一样。

需要知道各个运算符（算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符、条件运算符）的作用及其优先级。P42-50.

其他运算符：

new运算符：创建对象和调用构造函数。

is运算符：检查对象是否是某种类型。

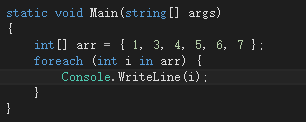
sizeof运算符（可以看做是一个函数）：取某个类型在本机所占的字节数。

typeof运算符（可以看做一个函数）：获取某个对象的类型。

**流程控制**：知道if、if-else、switch、while、do-while、for语句的使用及其坑。

比如do-while语句至少执行一次，for语句最好不要在语句内部改变计数器的值等等。

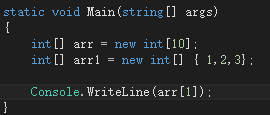
foreach：该流程控制语句的作用是从某一个可迭代对象中每次取出一个对象，并在循环体执行操作。



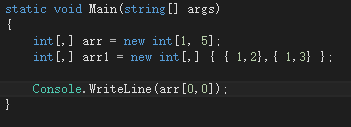
比如右边这段代码会把数组的所有元素按顺序输出。

**数组**：

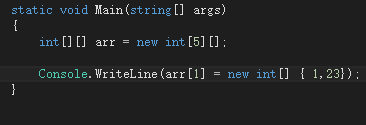
一维数组的声明及使用：



二维数组：



交错数组（数组类型的数组）：



数组也是一种引用类型。

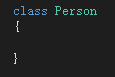
# 第三章

**类和对象**：

类是对具体事物的抽象，比如人类、动物类，这些都是类，它指的是某一类事物包含的一些共同特征，比如人类有身高体重等属性，有跑、坐等动作。

对象是类的实例，一个人，就是人类的一个实例，这个人拥有人类的全部属性及其行为。

**声明类**：



**修饰符**：

new：定义在嵌套类里，隐藏该类继承下来的同名成员。

internal：内部类，代表该类只能被当前的项目所使用。

public：公开类，访问不受限制。

protected：只能被它自己及其子类访问。

private：私有类，只能被当前类访问。

abstract：抽象类，只能被继承，不能被实例化。

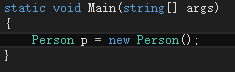
sealed：密封类，只能被实例化，不能被继承。

**类的成员**：

方法、属性、事件、构造函数、析构函数、索引、运算符。

**类的实例**：

创建一个对象：



**类与结构体**：

类是一种引用类型，结构体是值类型。

引用与值的区别：

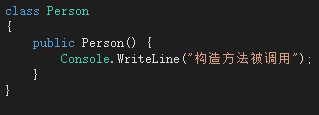
引用引用了内存当中的一份区域，如果a = new Person(),然后b=a，那么对b的所有操作，在a上都会有体现，而值类型则不一样，值类型每次被赋值时，都会产生一个副本，对一个副本的修改不会影响到另外一个。

**构造函数**：

特性：

1. 名称与类名相同。
2. 没有返回值类型。
3. 通常是public访问权限，如果为private的话，只能由该类自己调用。
4. 作用是完成对象的初始化。

可以通过base()来调用父类的构造器。



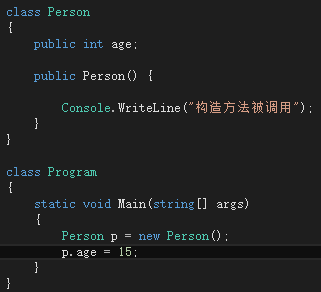
**析构函数**：

特性：没有修饰符也没有参数也没有返回值类型。名称为~加上类名。

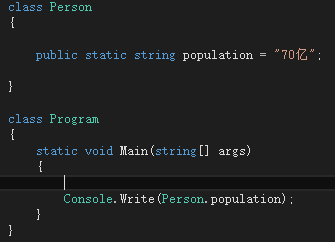
作用：由于C#是一种非手动管理内存的语言，所以由系统自己来管理内存，当对象占用的内存要被回收之前，垃圾回收器会调用这个析构方法。

**字段**：

与类或对象相关的变量。



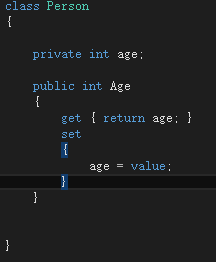
如果加上修饰符static，这个字段就叫做静态字段，代表这个字段属于这个类，只有通过类才能访问这个字段。



如果加上readonly，代表该字段只读，只能在赋值或者构造函数中赋值，其他地方不能修改。

**属性**：

可以看做是一组方法，该方法能赋值和取值：



加上static和字段，这个属性就属于类。

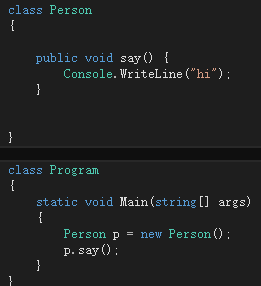
那么说了好几遍static这个关键字，也知道了这个关键字的意思是某个成员属于这个类，这又是什么意思。

属于这个类，打个比方，当前人类拥有人口数量为70亿，所以人口数量这个属性，就属于人类这个类，而不是人类的某一个实例人。

**方法**：

**方法的声明及使用**：

注意静态方法的使用。



**方法的参数类型**：

值参数：没有任何修饰符，当传入值参数时，程序将传入参数的一份副本。

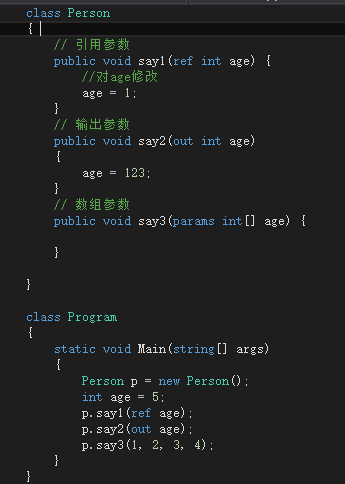
引用参数：ref修饰符，当传入引用参数时，程序将传入该参数的一个引用，所以，在方法内对形参（函数的参数）的修改，都会在实参（传入的参数）上表现出来。

输出参数：out修饰符，可以用于携带多个计算结果，与引用参数不同的是输出参数无需对变量进行初始化。

数组参数：

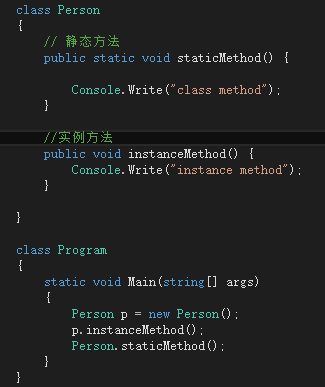
params关键字。

如果一个方法拥有数组参数，那么该参数必须是参数列表中的最后一个参数。



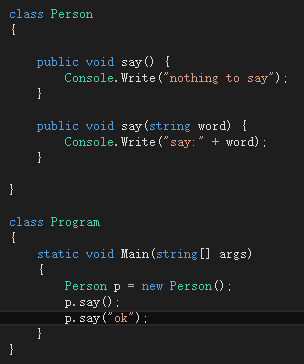
**静态方法与实例方法**：

静态方法是属于类的方法，只能由类进行调用。实例方法是属于对象的方法，只有对象才能调用。



**方法重载**：

所谓方法重载，就是指一个类内有多个同名的方法，但是参数列表（参数的个数、类型等不同）不同



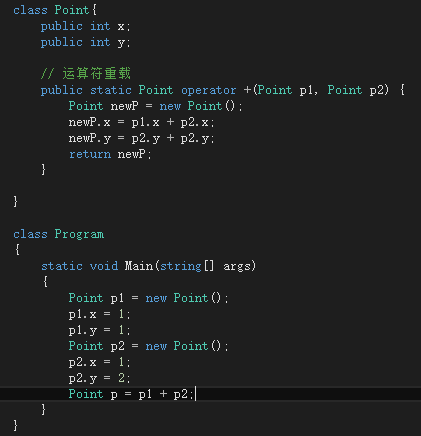
如上图所示，这样根据传入的参数列表不同，程序就会有不同的表现。

注意：方法重载重载的是参数列表，只要参数类型、长度不同，就可以看作是方法重载，返回值类型同不同无所谓。

**运算符重载**：

可以使用运算符重载，来使我们定义的类可以相加，或者进行相减等操作。

注意：运算符重载必须声明为static（该运算符属于类而非属于某个对象）。

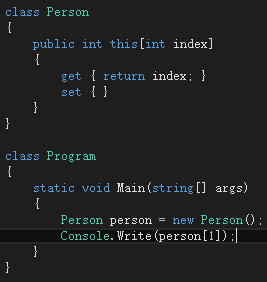


**this关键字：**

只能在构造函数或者实例方法中使用。代表当前对象。

**索引器**：

可以看做是一种运算符重载，是对[]运算符的重载。



# 第四章

**继承**：

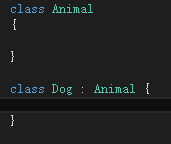
在已有类的基础上创建出新类。

概念：

子类（派生类、导出类）：主动去继承的那个类。

父类（超类，基类）：被继承的那个类。

语法:

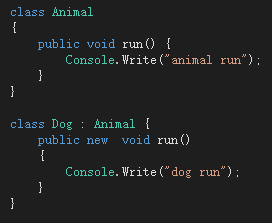


如果一个类被sealed修饰符修饰，代表该类是一个密封类（也称最终类），将无法被继承，所以abstract和sealed不能同时出现。

**隐藏父类方法**：

在子类当中，可以对父类方法进行隐藏，这样子类的方法就会覆盖父类的方法。

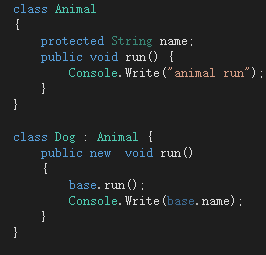
重写方法，加new修饰符



**base关键字**：

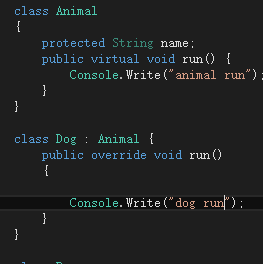
C#中可以使用base关键字来访问父类的成员，使用base的情况一是调用父类的方法，二是调用父类的成员变量。

要能想调用父类的成员，那么被调用成员的访问权限必须为public或者protected。



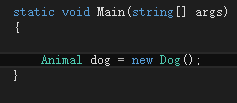
**重写**：

当父类的方法被virtual修饰时，表明该方法可以被子类使用override关键字重写：



**多态性**：

所谓多态，就是程序直到运行时，才确定对象的具体类型，比如有一个狗类，它继承了动物类，那么在运行时，将这个动物类对象替换成一个狗类对象，也可以替换成继承了动物类的其他动物类：



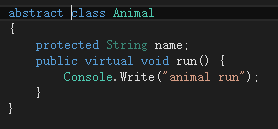
这叫做向上转型。

**抽象类和抽象方法**：

抽象，就是和具体相反，比如动物类，就可以看作是一种抽象类，因为它指的并不是某一种具体的类。

**声明抽象类**：

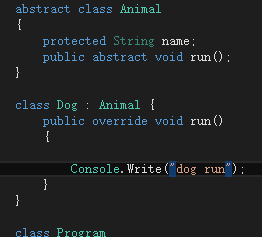
abstract关键字。



抽象类不能被实例化，也就是说在这里我们不能直接new Animal（）。

**抽象方法**：

抽象方法就是在抽象类中还未被实现的方法，就像Animal，因为不知道当前具体是什么样的Animal，肯定不知道是怎样run的，所以必须把run方法留给它的子类去实现：



一个有抽象方法的类必定是一个抽象类。但是一个抽象类不一定有抽象方法。

**静态类和静态方法**：

静态类就是加了static修饰符，在静态类中，不能有实例成员，只能有静态成员。

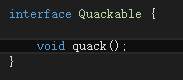
静态方法就是属于类的方法，只能通过类来调用。

# 第五章

接口：

可以与现实世界的接口类比，比如电脑的USB接口，不管是什么设备，只要该设备实现了USB接口，就能通过这个接口与计算机交换数据。

声明：

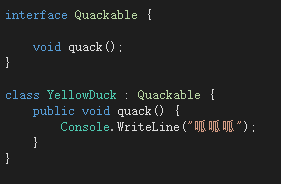


特性：

1. 由方法、属性、事件、索引器四种成员组合而成，没有任何具体实现。
2. 不能包含和对象有关的任何东西。
3. 接口成员自动就是公开权限的，不能使用任何访问修饰符。
4. 实现接口的类，必须实现该接口的所有成员。
5. 父类实现接口时，子类将继承所有实现。
6. 一个类可以实现多个接口。

总而言之，接口，就是一种定义，一种规约。

接口实现：



接口映射：

接口中的每一个成员在实现类都有与之相对应的映射。

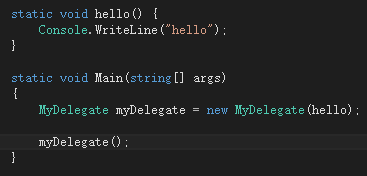
**委托**：

委托，就是可以封装函数，这样函数就可以像对象一样，变成一个可以被传递的引用。

委托的声明：

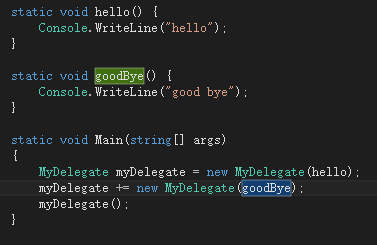


其中委托的参数列表与返回值都与被委托的函数一样。



可以这么理解，委托就是一个中间人，当创建委托的时候，向他传入一个函数名，这样就可以通过这个委托来调用函数。

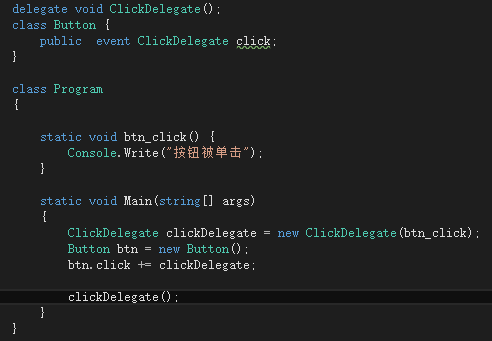
多播委托：

多播，委托一次性代理多个函数，这样只需一次调用，就能执行多个函数。 

**事件**：

定义方法：

1. 定义事件的委托。
2. 声明触发事件的类。
3. 订阅事件。



说明：事件也就是通过+=运算符来注册一个委托到事件，这样事件发生时，就会调用这个委托，进而调用委托所代理的函数。

# 第六章

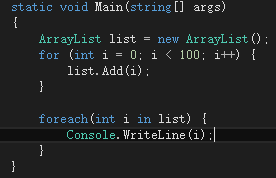
使用集合类，需要引用System.Collections命名空间。

ArrayList类：

对应数据结构当中的顺序表。

常用属性及方法参考书本146 147。

使用实例：

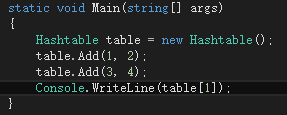


Hashtable类：

对应数据结构中的哈希表。

可以给它一个键，一个值，它的内部会维护这种键值对的关系，当需要一个对象的使用，可以通过键快速地获取到那个对象。

常用属性及方法参考148 149.

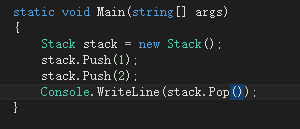


Stack类：

对应数据结构的栈。

常用属性及方法：151 152

后进先出：

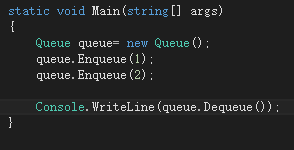


Queue类：

对应数据结构的队列。

先进先出。

常用属性及方法：153

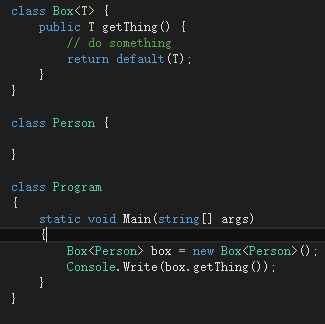


**泛型**：

泛型，就是一种类型参数，在创建某个类的对象时，可以通过传入一个类型参数，使代码更加灵活、安全。

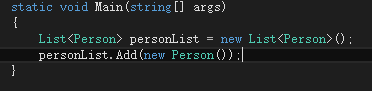
比如：有一个容器，理论上什么东西都能装。但是为了类型安全，避免出错，可以在创建容器的时候指定这个容器只能装什么，这就是泛型的一种用法。

创建一个可以装人的箱子：



List<T>类：

可以看做是一种能存放T类型对象的List。

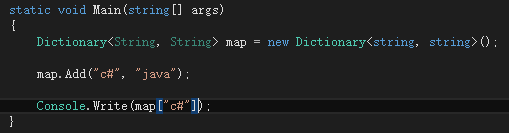


在这里，List只能被用来存放Person类的对象。

常用属性及方法：155 156。

Dictionary<K,V>类：

可以看做是一种只能存贮键为K类型，值为V类型的哈希表。



在这里，就创建了一个字符串映射字符串的哈希表。

常用属性及方法157 158.

# 第七章

**异常**：

异常，就是程序执行的不正常情况，比如一个长度只有2的顺序表，是不能获取它的下标为2的元素的，否则会产生一个下标越界异常。

为什么要有异常？异常机制可以让普通代码与异常处理代码分离开来。

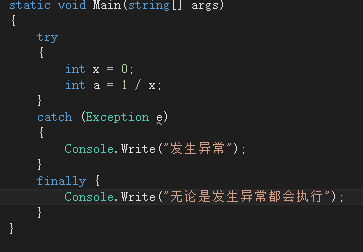
**异常控制机制**：

几个关键字：try：括号后的代码如果发生异常，就会交给catch关键字后的语句进行处理。

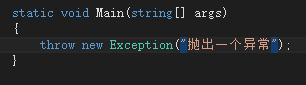
catch：后面紧跟着异常处理代码。

finally：这个关键字的意思是该关键字后的语句无论什么情况都会被执行，一般是用来进行关闭资源等操作。

throw：该关键字可以在抛出一个异常。



抛出一个异常：



.NET Framework中的异常：170.

# 第八章

详细内容略。。。

预习指南：

1. 了解有哪些控件。
2. 了解每个控件有哪些属性。
3. 了解每个控件有哪些事件。
4. 了解如何给控件添加事件。
5. 了解每个控件的常用属性的操作。