Unity 1143 C#编程-第二季-面向对象

出自SiKi学院 www.sikiedu.com SiKi老师

C#系列课程一共五季

第一季 - 编程基础

第二季 - LeetCode编程面试题

第三季 - 面向对象

第四季 - 常用类和高级语法

第五季 - 数据结构和算法

顺序修改如下

C#系列课程一共五季

第一季 - 编程基础

第二季 - 面向对象

    坦克大战

第三季 - 常用类和高级语法

第四季 - 数据结构和算法

第五季 - LeetCode编程面试题

推荐的C#书籍：

C#入门经典 第8版

C#高级编程(第11版)

怎么搜索程序方面的书籍

当当

亚马逊

开发工具：

Visual Studio 2019 Community社区版

调试和错误处理

错误（Bug）：

代码中难免存在错误，不论程序员多么优秀，程序总会出现问题，有些问题，比如变量名字写错，导致编译器无法编译（语法错误），有的时候我们的逻辑在某些方面有瑕疵，也会产生错误，这类错误成为语义错误（逻辑错误）。

我们接下来学习如何在程序出错之后，使用VS提供给我们的调试功能，找到错误的原因，修改代码。(调试)
以及学习c#中的错误处理技术，对可能发生错误的地方采取预防措施，并编写弹性代码来处理可能会发生的致命错误。(错误处理)

正常模式下的调试：

正常模式指的是不会影响程序的正常运行。

1,在VS中我们使用Console.Write(或者WriteLine)方法向控制台输出变量的值，通过这个我们可以查看变量的值是否符合我们的预期来调试错误。

2,在Unity中我们使用Debug.Log("") Debug.LogError("") Debug.LogWarn("")，向unity的Console窗口输出信息，帮助我们调试错误。

中断（Debug）模式下的调试：

中断模式指我们可以暂停程序的执行，然后查看程序中的状态，也可以让程序继续执行。

如何让程序中断？ 断点

断点是什么？
断点是源代码中自动进入中断模式的一个标记，当遇到断点的时候，程序会进入中断模式。

如何插入断点？
1，右击代码行，选择breakpoint(断点) -> insert breakpoint（插入断点）
2，光标定位到代码行，选择菜单上的Debug(调试)->Toggle Breakpoint(切换断点)
3，光标定位到代码行，按下F9键，在此按下F9是取消断点
4，在需要添加断点的行首位置，直接单击，再次单击取消断点

窗口 Breakpoints

我们可以通过 (调试-窗口-断点)，打开断点窗口，这个窗口显示了当前项目中添加了的所有的断点，我们可以在这里定位断点的位置，也可以去删除断点。

监视变量的内容

在中断模式下查看变量值最简单的方式，就是把鼠标指向源代码中的变量名，此时会出现一个工具提示，显示该变量的信息。

中断模式下的窗口（左下角），有三个选项卡
　错误列表　　-程序运行中发生的所有错误的列表
　局部变量　　-当前运行环境中所有的局部变量的值
　监视　 -监视某个变量的值的变化
在上面的几个窗口中不但可以观察变量值的变化，还可以直接去修改变量中存储的值

调用堆栈和即时窗口：

在中断模式下，可以在右下角看到调用堆栈和即时窗口

在调用堆栈窗口下我们可以观察到当前代码执行到哪一行了，并且可以看到这个代码的是被什么语句调用的
即时窗口我们可以在这里输入一些命令，查看变量的值，修改变量的值，可以输入表达式查看结果

单步执行代码：

在中断模式下我们可以单步执行代码，单步执行带有有两种 逐过程和逐语句，他们两个都是一条语句一跳语句的执行，区别在于逐过程遇到函数，不会进入函数内部，而把函数当成一条语句去执行。

错误处理(异常处理)

我们上面讨论了再开发过程中如何查找和修正错误，使这些错误不会再发布的代码中出现，但有时，我们知道可能会有错误发生，但不能100%的肯定他们不会发生，此时最好能预料到错误的发生，编写足够健壮的代码以处理这些错误，而不必中断程序的执行。

错误处理就是用于这个目的。下面学习异常和处理他们的方式。

异常：

异常是在运行期间代码中产生的错误。
示例：

```
int[] myArray = {1,2,3,4};
int myEle = myArray[4];//数组下标越界
```

运行到这里的时候，会出现异常，这个异常的定义已经在CLR中定义好了。如果我们不去处理这个异常，那么当异常发生的时候，程序会终止掉，然后异常后面的代码都无法执行。

异常处理(捕捉异常) try ... catch ... finally 语句

我们处理异常的语法结构如下(包含了三个关键字 try catch finally)

```
try{
...
}
catch( <exceptionType> e ){
...
}
finally{
}
```

其中catch块可以有0或者多个，finally可以有0或者1个
但是如果没有catch块，必须有finally块，没有finally块，必须有catch块，catch块和finally块可以同时存在

每个代码块的用法

try块包含了可能出现异常的代码(一条或者多条语句)

catch块用来捕捉异常，当代码发生异常，那么异常的类型和catch块中的类型一样的时候，就会执行该catch块，如果catch块的参数不写，表示发生任何异常都执行这个catch块

finally块包含了始终会执行的代码，不管有没有异常产生都会执行

编程题：

1、处理刚刚的下标越界异常。

2、让用户输入两个数字，用户可能会出入非数字类型，处理该异常，如果出现该异常就让用户重新输入，输出这两个数字的和

面向对象编程

什么是面向对象编程？

为什么使用面向对象编程？

为了让编程更加清晰，把程序中的功能进行模块化划分，每个模块提供特定的功能，而且每个模块都是孤立的，这种模块化编程提供了非常大的多样性，大大增加了重用代码的机会。
面向对象编程也叫做OOP编程
简单来说面向对象编程就是结构化编程，对程序中的变量结构划分，让编程更清晰。

注意：不要从字面意思理解面向对象编程

我们把类创建的变量叫做对象，那么如何创建类呢？

类是什么东西？

类实际上是创建对象的模板，每个对象都包含数据，并提供了处理和访问数据的方法。
类定义了类的每个对象（称为实例）可以包含什么数据和功能。

类的定义

类中的数据和函数称为类的成员
　数据成员
　函数成员
数据成员：
　数据成员是包含类的数据--字段，常量和事件的成员。
函数成员：
　函数成员提供了操作类中数据的某些功能。（方法，属性，构造方法和终结器（析构方法），运算符，和索引器）

类的字段和方法

字段的声明
    访问修饰符 类型 字段名称;
方法的声明
    访问修饰符 返回值类型 方法名称(参数){
        //方法体
    }

定义一个类来表示一个顾客

```
class Customer{
  public string name;
  public string address;
  public int age;
  public string buyTime;
  public void Show(){
    Console.WriteLine("名字:"+name);
    Console.WriteLine("年龄:"+age);
    Console.WriteLine("地址:"+address);
    Console.WriteLine("购买时间:"+buyTime);
  }
}
```

在这里我们定义了一个类，叫做Customer（顾客），里面包含了四个字段，存储了顾客的名字，年龄，地址和购买时间，包含了一个Show（）方法，用来输出自身的一些信息的，在这里Customer就是一个新的数据类型
类似于结构体的定义
struct Customer.....;

如何利用类创建对象

使用我们自定义的类声明的变量也叫做对象，这个过程也叫做实例化。

```
ClassName myClass = new ClassName();
```

其中ClassName是我们定义的类的名字，myClass是我们声明的变量（对象）的名字，后面的new是一个关键字，使用new 加上类型名（）表示对该对象进行构造，如果不进行构造的话，这个对象是无法使用的。

类的定义和声明

定义一个车辆（Vehicle）类，具有Run、Stop等方法，具有Speed（速度）、MaxSpeed（最大速度）、Weight（重量）等域（也叫做字段）。
使用这个类声明一个变量（对象）

类的定义和声明

定义一个向量（Vector3）类，里面有x，y，z三个字段，有取得长度的方法，有设置属性（Set）的方法
使用这个类声明一个变量（对象）

构造函数

我们构造对象的时候，对象的初始化过程是自动完成的，但是在初始化对象的过程中有的时候需要做一些额外的工作，例如需要初始化对象存储的数据，构造函数就是用于初始化数据的函数。
声明基本的构造函数的语法就是声明一个和所在类同名的方法，但是该方法没有返回类型。

```
public class MyClass{
  public MyClass(){
    这个构造函数的函数体
  }
}
```

当我们使用new关键字创建类的时候，就会调用构造方法。
我们一般会使用构造方法进行初始化数据的一些操作。
构造函数可以进行重载，跟普通函数重载是一样的规则
注意：
　　当我们不写，任何构造函数的时候，编译器会提供给我们一个默认的 无参的构造函数，但是如果我们定义了一个或者多个构造函数，编译器就不会再提供默认的构造函数

属性的定义

属性的定义结构：

```
public int MyIntProp{
  get{
  // get code
  }
  set{
  //set code
  }
}
```

1，定义属性需要名字和类型
2，属性包含两个块 get块和set块
3，访问属性和访问字段一样，当取得属性的值的时候，就会调用属性中的get块，所以get块，类型需要一个返回值就是属性的类型；当我们去给属性设置值的时候，就会调用属性中的set块，我们可以在set块中通过value访问到我们设置的值。

通过属性来访问字段

我们习惯上把字段设置为私有的，这样外界不能修改字段的值，然后我们可以通过定义属性来设置和取得字段中的值。

```
private int age;
public int Age{//习惯上属性大写 字段小写
```

```
  set{
    if(value<0)return;
    age = value;
  }
  get{
    return age;
  }
}
```

1、设置属性的只读或者只写

```
private string name;
public string name{
  get{
    return name;
  }
}
```

属性可以值只提供一个set块或者get块

2、属性的访问修饰符

```
public string name{
  get{
    return name;
  }
  private set{
    name = value;
  }
}
```

3、自动实现的属性

public int Age{get;set;}
编译器会自动创建private int age属性

匿名类型

我们创建变量（对象的时候），必须指定类型，其实我们也可以不去指定类型，这个就是匿名类型，我们可以使用var声明一个匿名类型。
使用var声明的匿名类型，当初始化的时候，这个变量的类型就被确定下来，并且以后不可以修改。

var var1 = 34;

程序内存区域：堆 栈 静态存储区

程序所有的数据，也就是所有的变量，都是存储在内存中的。

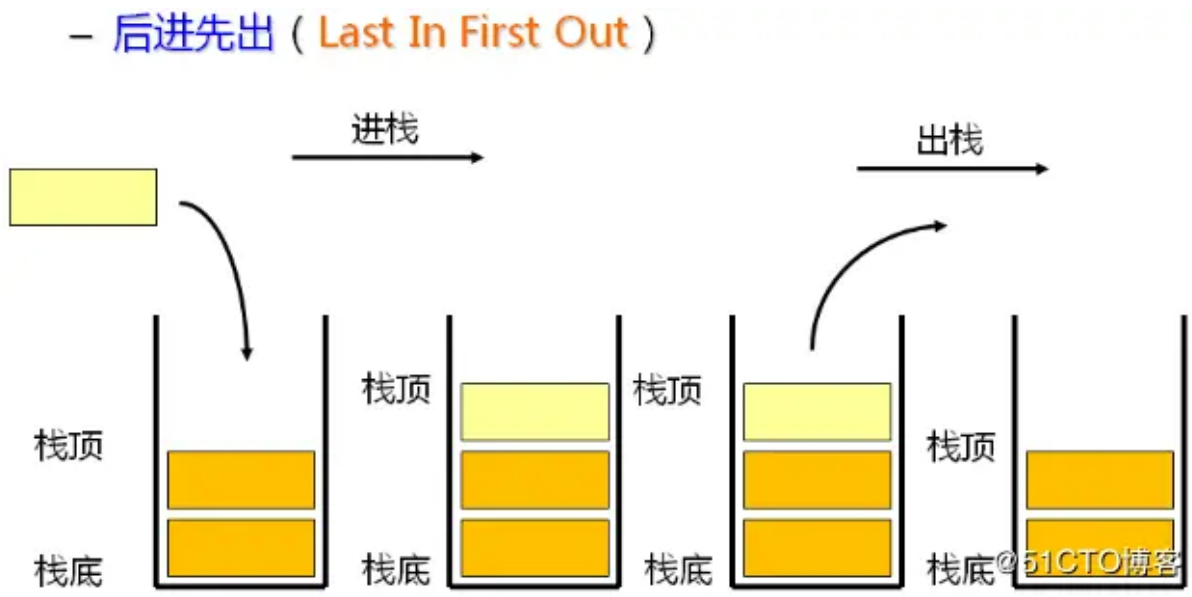
栈空间比较小，但是读取速度快
堆空间比较大，但是读取速度慢

商店和仓库

栈

栈的特征：

数据只能从栈的顶端插入和删除
把数据放入栈顶称为入栈（push）
从栈顶删除数据称为出栈（pop）

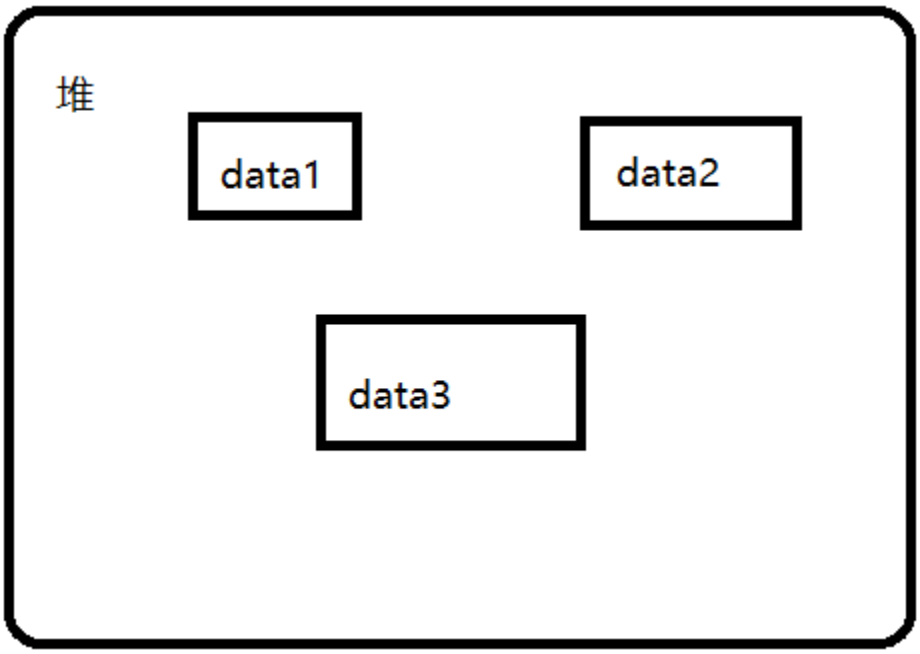


内存中的栈，是由系统管理（.Net框架）。

我们之后会在数据结构中学习写一个自己的栈结构，保存数据。

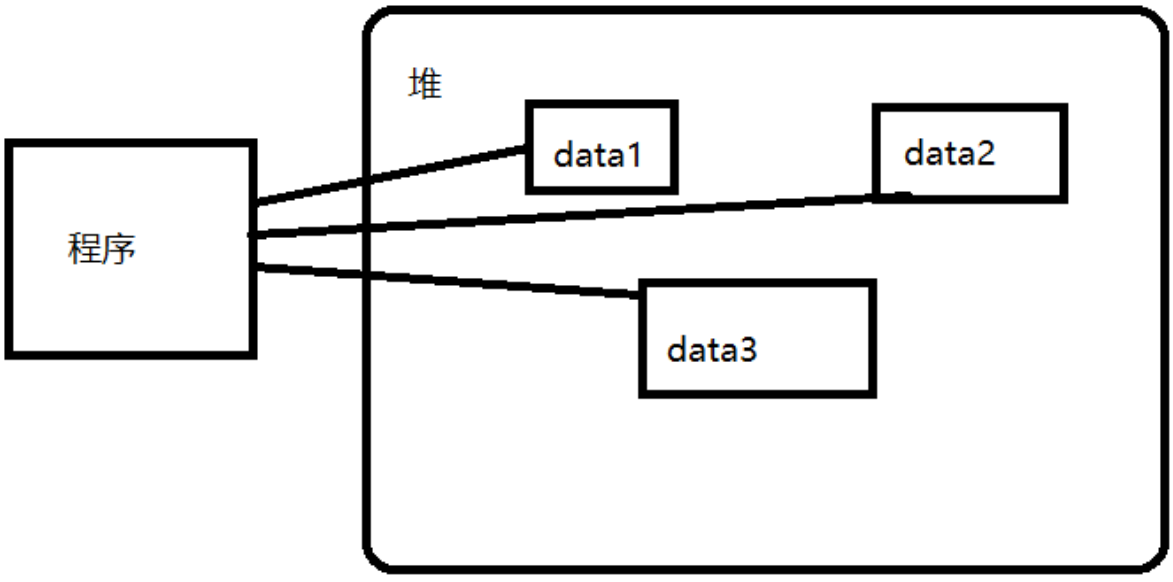
堆

堆是一块内存区域，与栈不同，堆里的内存能够以任意顺序存入和移除



GC Garbage Collector垃圾回收器

CLR的GC就是内存管理机制，我们写程序不需要关心内存的使用，因为这些都是CLR帮我们做了。



这里的某些概念，只需要了解和知道的，不需要知道内在核心原理，我们知道汽车的引擎是汽车的核心动力来源，我们不需要知道引擎是怎么造。

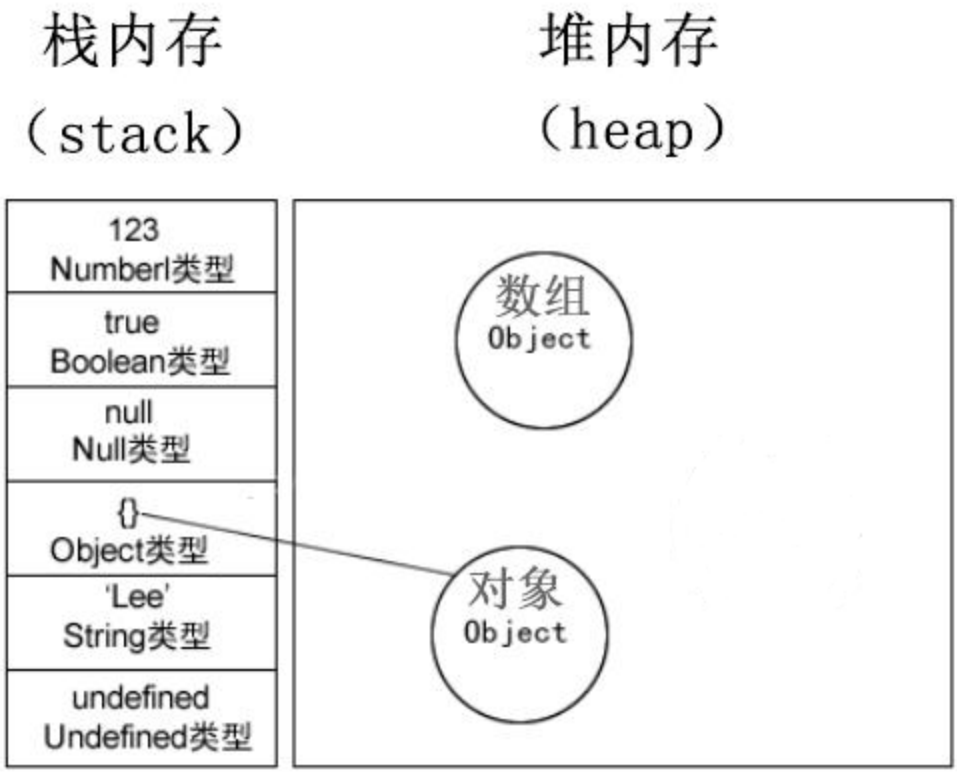
同样，上述是编译器开发者的事情。

值类型和引用类型

类型被分为两种：值类型(整数，bool struct char 小数)和引用类型（string 数组 自定义的类，内置的类）。

值类型只需要一段单独的内存，用于存储实际的数据，（单独定义的时候放在栈中）
引用类型需要两段内存

第一段存储实际的数据，它总是位于堆中
第二段是一个引用，指向数据在堆中的存放位置



注意：

当我们使用引用类型赋值的
时候，其实是赋值的引用类型的引用
如果数组是一个值类型的数组，那么数组中直接存储值，如果是一个引用类型的数组（数组中存储的是引用类型），那么数组中存储的是引用（内存地址）

面向对象-继承

继承是什么？

在上一节课中学习了如何定义类，用类当做模板来声明我们的数据。
很多类中有相似的数据，比如在一个游戏中，有Boss类，小怪类Enemy，这些类他们有很多相同的属性，也有不同的，这个时候我们可以使用继承来让这两个类继承自同一个类。

继承的类型

实现继承：

> 表示一个类型派生于一个基类型,它拥有该基类型的所有成员字段和函数。 在实现继承中,派生类型采用基类型的每个函数的实现代码,除非在派生类型的定义中指定重写某个函数的实现代码。 在需要给现有的类型添加功能,或许多相关的类型共享一组重要的公共功能时,这种类型的继承非常有用。

接口继承：

> 表示一个类型只继承了函数的签名,没有继承任何实现代码。 在需要指定该类型具有某些可用的特性时,最好使用这种类型的继承。

多重继承

一些语言(C++)支持所谓的 “多重继承”,即一个类派生自多个类。 使用多重继承的优点是有争议的:一方面,毫无疑问,可 以使用多重继承编写非常复杂、 但很紧凑的代码,。另一方面,使用多重实现继承的代码常常很难理解和调试。 如前所述,简化健壮代码的编写工作是开发 C#的重要设计 目标。 因此,C#不支持多重实现继承。 而 C#允许类型派生自多个接

口— — 多重接口继承。 这说明,C#类可以派生自另一个类和任意多个接口。更准确地说,System.Object 是一个公共的基类,所 以每个 C#(除了Object类之外)都有一个基类,还可以有任意多个基接 口。

实现继承

如果要声明派生自另一个类的一个类,就可以使用下面的语法:

```
class MyDerivedClass : MyBaseclass
{
// functions and data members here
}
```

如果类(或 结构)也 派生 自接 口,则用逗号分隔列表中的基类和接 口:

```
public class MyDerivedClass: MyBaseClass , IInterface1 , IInterface2
{
// etc.
}
```

案例实现

基类敌人类(hp speed 方法 ai move)
派生出来两个类

- boss类
- type1enemy类
- type2enemy类

this和base关键字

this（base）作用：

1、IDE给提示

2、区分局部变量和字段

this可以访问当前类中定义的字段,属性和方法，有没有this都可以访问，有this可以让IDE-VS编译器给出提示，另外当方法的参数跟字段重名的时候，使用this可以表明访问的是类中的字段，base可以调用父类中的公有方法和字段,有没有base都可以访问，但是加上base.IED工具会给出提示，把所有可以调用的字段和方法罗列出来方便选择

虚方法

把一个基类函数声明为virtual,就可以在任何派生类中重写该函数:

```
class MyBaseClass{
  public virtual string VirtualMethod(){
    return "Method is called in base class";
  }
}
```

在派生类中重写另外一个函数时，要使用override关键字显示声明

```
class MyDerivedClass:MyBaseClass{
  public override string VirtualMethod(){
    return "Method is called in derivedclass.";
  }
}
```

我们在子类里面重写虚函数之后，不管在哪里调用都是调用重写之后的方法

隐藏方法

如果签名相同的方法在基类和派生类中都进行了声明，但是该方法没有分别声明为virtual和override，派生类就会隐藏基类方法。（要使用new关键字进行声明）
基类

```
class MyBaseClass{
  public int MyMethod(){
```

```
  }
}
```

派生类(在派生类中把基类同名的方法隐藏掉了)

```
class MyDerivedClass :MyBaseClass{
  public new void MyMethod() {
  }
}
```

抽象类

C#允许把类和函数声明为 abstract。 抽象类不能实例化,抽象类可以包含普通函数和抽象函数，抽象函数就是只有函数定义没有函数体。 显然,抽象函数本身也是虚拟的Virtual(只有函数定义，没有函数体实现)。
类是一个模板，那么抽象类就是一个不完整的模板，我们不能使用不完整的模板去构造对象。

```
abstract class Building{
  public abstract decimal CalculateHeatingCost();
}
```

密封类和密封方法

C#允许把类和方法声明为 sealed。 对于类 ,这表示不能继承该类；对于方法表示不能重写该方法。

```
sealed FinalClass
{
// etc
}
```

什么时候使用 密封类和密封方法？

防止重写某些类导致代码混乱
商业原因

派生类（子类）的构造函数

1,在子类中调用父类的默认构造函数（无参）（会先调用父类的，然后是子类的）

```
public class MyDerivedClass{
  public MyDerivedClass():base(){
    //do something
  }
}
```

在这里 ：base()可以直接不写，因为默认会调用父类中的默认构造函数
2，调用有参数的构造函数

```
public class MyDerivedClass{
  public MyDerivedClass(string name):base(name){
    //do something
  }
}
```

修饰符

修饰符，用来类型或者成员的关键字。修饰符可以指定方法的可见性。

- public: 同一程序集(DLL或EXE)中的任何其他代码或引用该程序集的其他程序集都可以访问该类型或成员。
- private: 只有同一类或结构中的代码可以访问该类型或成员。
- protected: 只有同一类或结构或者此类的派生类中的代码才可以访问该类型或成员。
- internal: 同一程序集中的任何代码都可以访问该类型或成员，但的代码不可以。
- protected internal: 在一程序集中，protected internal体现的是internal的性质；在其他程序集中，protected internal体现的是protected的性质。

public 和private修饰字段和方法的时候，表示该字段或者方法能不能通过对象去访问，只有public的才可以通过对象访问，private（私有的）只能在类模板内部访问。

protected 保护的，当没有继承的时候，它的作用和private是一样的，当有继承的时候，protected表示可以被子类访问的字段或者方法

类的修饰符

public class ...
class ...
前者可以在别的项目下访问，后者不行

其他修饰符

new

隐藏继承的成员

abstract

使用abstract修饰的类为**抽象类**，抽象类只能是其他类的基类，不能与sealed、static一起使用。

abstract可以修饰抽象类中的方法或属性，此时，方法或属性不能包含实现，且访问级别不能为私有。

抽象类不能被实例化。

sealed

使用sealed修饰的类为**密封类**，密封类无法被继承，不能和abstract、static一起使用。

当sealed用于方法或属性时，必须始终与override一起使用。

static

使用static修饰的类为**静态类**，静态类所有成员都必须是静态的，不能与abstract、sealed一起使用。

static可以修饰方法、字段、属性或事件，始终通过类名而不是实例名称访问静态成员，静态字段只有一个副本。

静态类不能被实例化。

const

使用const关键字来声明某个常量字段或常量局部变量，必须在声明常量时赋初值。

不能与static一起使用，常量默认是static的，常量字段只有一个副本。

readonly

使用readonly关键字来声明只读字段。

只读字段可以在声明或构造函数中初始化，每个类或结构的实例都有一个独立的副本。

可以与static一起使用，声明静态只读字段。

静态只读字段可以在声明或静态构造函数中初始化，静态常量字段只有一个副本。

virtual

virtual关键字用于修饰方法、属性、索引器或事件声明，并使它们可以在派生类中被重写。

默认情况下，方法是非虚拟的。 不能重写非虚方法。

virtual修饰符不能与static、abstract、private或override修饰符一起使用。

override

要扩展或修改继承的方法、属性、索引器或事件的抽象实现或虚实现，必须使用override修饰符。

重写的成员必须是virtual、abstract或override的。

引用自：https://www.cnblogs.com/zhouzl/archive/2019/04/13/10701476.html

定义和实现接口

定义接口(飞翔功能)

```
public interface IFlyHandler{
  public void Fly();
}
```

实现接口

```
public class Type1Enemy:IFlyHandler{
}
```

USB接口，耳机接口。只提供了接口，没有提供实现，让其他开发商实现。

定义一个接口在语法上跟定义一个抽象类完全相同，但不允许提供接口中任何成员的实现方式，一般情况下，接口只能包含方法，属性，索引器和事件的声明。
接口不能有构造函数，也不能有字段，接口也不允许运算符重载。
接口定义中不允许声明成员的修饰符，接口成员都是公有的

派生的接口

接口可以彼此继承，其方式和类的继承方式相同

```
public interface A{
  void Method1();
}
public interface B:A{
  void Method2();
}
```

其他

索引器
运算符重载

面向对象 — 编程题

集合类 列表List

当我们有很多类型一样的数据的时候，前面我们一般使用数组来进行管理，但是这样有个缺点就是数组的大小是固定的。如果我们很多类型一样的数据，比如游戏得分，我们可以集合类来进行管理，比如列表List，我们可以使用列表List很方便的添加数据，删除数据还有其他对数据的操作。

列表List的创建和使用

1,创建列表(列表可以存储任何类型的数据，在创建列表对象的时候首先要指定你要创建的这个列表要存储什么类型的)（泛型）

```
List<int> scoreList = new List<int>();
new List<int>(){1,2,3}
new List<string>(){"one","two"}
var scoreList = new List<int>();
```

2,往列表中插入数据

```
scoreList.Add(12);
scoreList.Add(45);
```

3,如何取得列表中的数据？列表中的数据跟数组有点相似，索引从0开始 ，可以通过索引来访问
scoreList[0] //访问添加到列表中的第一个数据

关于列表的更多内容

1,列表内部数据是使用数组进行的存储，一个空的列表内部会有一个长度为0的数组，当给列表中添加元素的时候，列表的容量会扩大为4，如果添加第5个的时候，列表的大小会重新设置为8，如果添加第9个元素，列表容量会扩大为16，依次增加。当列表的中的容量发生改变的时候，它会创建一个新的数组，使用Array.Copy()方法将旧数组中的元素复制到新数组中。为了节省时间，如果事先知道要存储的数据的个数，就可以利用列表的构造函数指定列表的容量大小，比如下面的
List<int> intlist = new List<int>(10);创建了一个初始容量为10的列表，当容量不够用的时候，每次都会按照原来容量的2倍进行扩容。
我们可以通过Capacity属性获取和设置容量
intList.Capacity = 100;
2,注意容量和列表中元素个数的区别，容量是列表中用于存储数据的数组的长度通过Ｃａｐａｃｉｔｙ获取，列表中的元素是我们添加进去需要管理的数据，通过Count获取

列表的遍历

遍历列表有两种方式：

1,for循环，遍历所有的索引，通过索引访问列表中的元素
```
for(int i=0;i<list.Count;i++){
//循环体list[i]
}
```

2,foreach遍历
```
foreach(int temp in list){ //依次取得list中的每一个元素赋值给temp，并执行循环体
//循环体 temp
}
```

操作列表的属性和方法

1,Capacity获取容量大小
2,Add()方法添加元素

3,Insert()方法插入元素
4,[index]访问元素
5,Count属性访问元素个数
6,RemoveAt()方法移除指定位置的元素
7,IndexOf()方法取得一个元素所在列表中的索引位置
LastIndexOf()上面的方法是从前往后搜索，这个是从后往前搜索，搜索到满足条件的就停止
上面的两个方法，如果没有找到指定元素就返回-1
8,Sort()对列表中是元素进行从小到大排序

泛型

泛型是什么？

通过参数化类型来实现在同一份代码上操作多种数据类型。利用“参数化类型”将类型抽象化，从而实现灵活的复用。

泛型类定义

定义一个泛型类就是指的是，定义一个类，这个类中某些字段的类型是不确定的，这些类型可以在类构造的时候确定下来,举例:
创建一个类处理int类型和double类型的相加

```
class ClassA<T>{
  private T a;
  private T b;
  public ClassA(T a,T b){
    this.a = a ;this.b = b;
  }
  public T GetSum(){
    return a+“”+b;
  }
}
```

泛型方法

定义泛型方法就是定义一个方法，这个方法的参数的类型可以是不确定的，当调用这个方法的时候再去确定方法的参数的类型。
实现任意类型组拼成字符串的方法

```
public static T GetSum<T>(T a,T b){
  return a+""+b;
}
GetSum<int>(23,12);
GetSum<double>(23.2,12);
```

使用泛型和索引器来实现一个我们自己的集合类MyList

有下面的方法和属性
1,Capacity获取容量大小
2,Add()方法添加元素
3,Insert()方法插入元素
4,[index]访问元素(索引器)
5,Count属性访问元素个数
6,RemoveAt()方法移除指定位置的元素
7,IndexOf()方法取得一个元素所在列表中的索引位置
LastIndexOf()上面的方法是从前往后搜索，这个是从后往前搜索，搜索到满足条件的就停止
上面的两个方法，如果没有找到指定元素就返回-1
8,Sort()对列表中是元素进行从小到大排序
索引器：通过[index]这种形式去访问数据，就是索引器

计划一个游戏案例