接口

接口是除了虚方法、抽象类外的第三种实现多态的方法

> 什么时候用接口?
> 答：1).所有对象都具有一个相同行为，却无法抽象出一个共同的父类时
> 2).当子类想实现多种功能时，却不能继承多个有那些功能的父类时(继承有单根性)
>
> 例如：麻雀会飞，鹦鹉会飞，鸵鸟不会飞，企鹅不会飞，直升飞机会飞
> 这时无法抽象出一个具有飞行能力的鸟类,因为鸵鸟和企鹅都是鸟却不会飞，直升飞机不是鸟却会飞
>
> 所以这时用接口方便，定义一个IFlyable接口，所有会飞的都继承这个接口，鸵鸟和企鹅不会飞就不继承

零、命名规则： I + ... + able;如：IFlyable 若加able不好读可以不强加
一、接口是一种规范、协议、能力
二、接口可以继承接口(可以多继承),但接口不能继承普通类
三、只要一个类继承了一个接口，就必须实现这个接口内的所有成员

```
//二、三举例:
public interface M1{
    void M1Test();
}
public interface M2{
    void M2Test();
}
public interface M3{
    void M3Test();
}
public interface SuperInterface : M1,M2,M3{
    //接口可以继承接口，但这两个接口的方法都得交给子类实现
    void SuperTest(string name);
}

public class Car : SuperInterface{
    //普通子类，一次实现全部接口的全部成员，包括接口的接口
    public void M1Test(){}
    public void M2Test(){}
    public void M3Test(){}
    public void SuperTest(string name){}
}
```

四、抽象类也可以继承接口，但接口的实现只能交给他的子类
五、接口可以有构造函数，但不能被实例化(类似于抽象类，不能被new对象)
六、接口中的成员不允许添加访问修饰符(没有访问修饰符，接口中默认就是public且不能修改)
七、接口中只能有方法、自动属性、索引器(还没学)，但自动属性和索引器本质上还是方法, 因此本质上接口中

只能有方法
八、接口中的成员不能有任何实现，方法不能有方法体
九、一个类可以同时继承一个父类，实现多个接口。若一个类同时继承了父类A，实现了接口I，那语法上A必须写在I的前面。 例：class MyClass : A , I {} 因为类是单继承的

```
    //四 ~ 九举例
    public interface IFlyable{
        //接口中只能有方法、自动属性、索引器(还没学)
        //但自动属性和索引器本质上还是方法
        //因此接口中只能有方法

        //接口中不允许有访问修饰符
        void Fly();
    }
    public interface IRunable{
        void Run();
    }
    public interface {
        void Eat();
    }

    //Person为抽象类，继承两个接口
    public abstract class Person : IFlyable , IRunable{
        //这里没有abstract，直接实现IFlyable中的Fly方法
        public void Fly(){
            System.Console.WriteLine("人类在飞");
        }
        //这里有abstract修饰，意思是将IRunable中的Run方法向下交给子类实现
        public abstract void Run();
    }

    //继承人类同时又多继承了一个“吃”接口
    public class Student : Person ,IEatable{    //语法上必须把父类写在接口前面，因为父类只能有一个，接口可以有无数个

        //往上溯源，Student一共继承了Person一个父类，IFlyable，IRunable，IEatable三个接口

        //由于父类中已实现Fly方法，因此这里只需要再实现Run和Eat方法即可
        public void Eat(){
            System.Console.WriteLine("学生在吃饭");
        }
        public override void Run()
        {
            System.Console.WriteLine("学生在跑");
        }
    }

    class MainFunction{
        static void Main(string[] args){
            Student stu = new Student();
            stu.Eat();
```

```
            stu.Run();
            stu.Fly();  //虽然student自己没有Fly方法，但可以调用父类Person中已实现的
Fly方法
            //结果为：
            //学生在吃饭
            //学生在跑
            //人类在飞

            //错误写法，接口不能被实例化
            //IFlyable fly = new IFlyable()

            //但可以实例化一个子类，然后里氏转换(赋值)给接口
            IFlyable fly = new Bird();
            fly.Fly();
        }
    }
```

十、显式实现接口的目的，解决方法的重名问题 什么时候显式地去实现接口 当继承的接口中的方法和参数一模一样的时候，要使用显式地实现接口

```
    //例：IFlyable里有Fly,Bird里也有Fly，当Bird继承接口时两个Fly会重名
    public interface IFlyable{
        void Fly();
    }
    public class Bird : IFlyable{
        //Bird自己的Fly
        public void Fly(){
            System.Console.WriteLine("鸟在飞");
        }

        //IFlyable接口里的Fly
        //使用void IFlyable.Fly()来显式调用，避免重名覆盖
        void IFlyable.Fly(){
            System.Console.WriteLine("接口里的飞");
        }
        //注：这里不能加访问修饰符
        //又因为这是在一个类内部，因为默认为private
        //外部能调用到是因为程序实际调用的是接口里的Fly(),而接口里的Fly()是public，这里
只是具体实现Fly()方法。
    }
    class MainFunction{
        static void Main(string[] args){
            //想要调用哪个Fly()就用哪个类的对象来调

            IFlyable fly = new Bird();//new Person();
            fly.Fly();      //IFlyable接口里的Fly()

            Bird bird = new Bird();
            bird.Fly();     //Bird自己的的Fly()
        }
    }
```

补充：普通属性与自动属性的区别

接口中不能存在普通属性，但可以存在自动属性

```
    public class Person{
        //普通属性:有字段，也有方法体
        private string _name;   //普通属性的字段
        public string Name
        {
            get
            {
                return this._name;  //普通属性的方法体
            }
            set
            {
                this._name = value; //普通属性的方法体
            }
        }

        //自动属性，不需要字段，也没有方法体
        public int Age
        {
            get;
            set;
        }

        //虽然我们在写自动属性时没有字段，但在编译过程中程序会自动生成一个私有字段。所以自动属性除了无法在属性内做限定外(可以在构造函数中限定)，其他都与普通属性相同

        //因为自动属性既无字段又无方法体的特性，所以它可以存在于接口内(去掉访问修饰符)
        public interface IFlyable{
            int Age { get; set; }
        }
    }
```