多态

1.减少代码冗余
2.屏蔽各个子类间的差异，写出通用的代码

实现多态的方法

1.1 虚方法

注：子类可以直接调用父类的虚方法，也可以重写后调自己的 步骤:

1. ： 将父类的方法标记为虚方法，使用关键字virtual
2. ： 将子类的方法重写，使用关键字overider

```
    //父类：Person
    //子类：Chinese、Japanese、Korea、American均继承于Person
    //父类中有SayHello方法，子类中也都有各自的SayHello方法

    Chinese cn1 = new Chinese("韩梅梅");
    Chinese cn2 = new Chinese("李雷");
    Japanese j1 = new Japanese("树下君");
    Japanese j2 = new Japanese("井边子");
    Korea k1 = new Korea("金秀贤");
    Korea k2 = new Korea("金贤秀");
    American a1 = new American("科比布莱恩特");
    American a2 = new American("奥尼尔");
    Person p = new Person("北京智人");      //与抽象类的不同之处：这里可以实例化父类对
象
    Person[] pers = {cn1,cn2,j1,j2,k1,k1,a1,a2,p};


    //1.不使用多态调用各子类的SayHello方法：
    for(int i = 0;i < pers.Length;i++){
        //需要判断各元素的类型然后强转成对应子类，才能调用子类的方法
        if(pers[i] is Chinese){
            ((Chinese)pers[i]).SayHello();
        }else if(pers[i] is Japanese){
            ((Japanese)pers[i]).SayHello();
        }else if(pers[i] is Korea){
            ((Korea)pers[i]).SayHello();
        }else{
            ((American)pers[i]).SayHello();
        }
    }

    //2.使用多态
    // 1).用Virtual将父类中的方法标记为虚方法
    public virtual void SayHello(){
        System.Console.WriteLine("我是人类，我叫： " + this.Name);
    }
```

```
    // 2).用overrider重写子类中的同名方法
    public override void SayHello(){
        System.Console.WriteLine("我是中国人，我叫：" + this.Name);
    }
    ......
    // 3).直接调用
    for(int i = 0;i < pers.Length;i++){
        pers[i].SayHello();
    }
    //两种方法结果为：
    //我是中国人，我叫：韩梅梅
    //我是中国人，我叫：李雷
    //我是脚盆国人，我叫：树下君
    //我是脚盆国人，我叫：井边子
    //我是棒子思密达，我叫：金秀贤
    //我是棒子思密达，我叫：金秀贤
    //我是米国人，我叫：科比布莱恩特
    //我是米国人，我叫：奥尼尔
    //我是人类，我叫：北京智人        //可见，父类的方法也可以调用，不影响
    //我是人类，我叫：北京智人
```

1.2 虚方法练习

```
    //练习
    //真的鸭子嘎嘎叫 木头鸭子吱吱叫 橡皮鸭子唧唧叫
    public class RealDuck{
        public virtual void Bark(){
            System.Console.WriteLine("真的鸭子嘎嘎叫");
        }
    }
    public class WoodDuck : RealDuck{
        public override void Bark(){
            System.Console.WriteLine("木头鸭子吱吱叫");
        }
    }
    public class RubberDuck : RealDuck{
        public override void Bark(){
            System.Console.WriteLine("橡皮鸭子唧唧叫");
        }
    }
    static void Main(string[] args){
        //真的鸭子嘎嘎叫 木头鸭子吱吱叫 橡皮鸭子唧唧叫
        RealDuck rd = new RealDuck();
        WoodDuck wd = new WoodDuck();
        RubberDuck rud = new RubberDuck();
        RealDuck[] ducks = {rd,wd,rud};
        for (int i = 0; i < ducks.Length; i++)
        {
            ducks[i].Bark();
        }
    }
```

2.1抽象类

当父类中的方法不知道怎样实现时，可以将父类写成抽象类，将方法写成抽象方法
使用abstract标记抽象类和抽象方法，在子类中使用override重写父类的抽象方法

注:
1). 抽象类中既可以有抽象成员，也可以有非抽象成员(包括虚方法)
1.1). 子类继承父类时，必须立刻实现父类的所有抽象成员（除非子类自己也是个抽象类)
1.2). 子类可以不实现父类的非抽象成员

2). 抽象方法不能有方法体，因此没有大括号；如public abstract void Bark( )
2.1). 如果父类的抽象方法中有返回值和参数，那其子类在重写时必须传入对应的参数，必须返回对应的返回值（注意这比一般的重写规则要更严格)

3). 抽象类有构造函数，但是都不能被实例化；即不能Animal a = new Animal()
3.1). 因为抽象类不能实例化，所以他调不了自己的非抽象成员（其抽象成员甚至都没有实现，所以更调不了) 3.2). 但他的非抽象成员仍然可以通过继承给子类使用，因此不是没有意义

8). 抽象成员只能写在抽象类中
9). 抽象成员的访问修饰符不能是private（因为写了private就不能被子类访问，就不能被重写实现)

```
    public abstract class Animal{
        //所有动物都会叫，而且叫的方法都不同，因此父类Animal中的Bark方法无法具体地实现，
应该写成抽象类，抽象方法

        //注:抽象方法不能有方法体，因此没有大括号{}
        public abstract void Bark();
    }
    public class Dog : Animal{
        //子类继承具有抽象方法的父类时，必须实现父类的抽象方法
        public override void Bark()
        {
            System.Console.WriteLine("狗狗汪汪汪地叫");
        }
    }

    static void Main(string[] args){
            //注：抽象类和接口都不能实例化
            //即不能Animal a = new Animal();

            Animal a = new Dog();   //创建一个子类对象赋值给父类

            //这里a是子类(Dog)对象，但表现为父类(Animal)类型
            //所以调用的方法是父类的方法
            //不过由于子类重写了父类的方法，因此最终调用的还是子类的方法
            a.Bark();   //结果为：狗狗汪汪汪地叫
        }
```

2.2 抽象类练习

```
//用多态来实现将U盘或者MP3插到电脑上进行读写数据
//MP3自己还有一个播放音乐功能

//定义一个抽象类：MobileStorage，U盘、MP3继承该类
public abstract class MobileStorage{
    public abstract void Read();
    public abstract void Write();
}
public class USBFlashDisk : MobileStorage{
    public override void Read()
    {
        System.Console.WriteLine("U盘在读取数据");
    }
    public override void Write()
    {
        System.Console.WriteLine("U盘在写入数据");
    }
}
public class MP3 : MobileStorage{
    public override void Read()
    {
        System.Console.WriteLine("MP3在读取数据");
    }
    public override void Write()
    {
        System.Console.WriteLine("MP3在写入数据");
    }
    public void PlayMusic(){
        System.Console.WriteLine("MP3在播放歌曲");
    }
}
//定义一个普通类：Computer，内含读写方法，调用MobileStorage的读写方法
//到时候直接用Computer即可实现子类的读写方法
public class Computer {
    private MobileStorage _ms;
    public MobileStorage MS { get; set; }
    public Computer(MobileStorage ms){
        this.MS = ms;
    }
    public void CpuRead(){
        MS.Read();
    }
    public void CpuWrite(){
        MS.Write();
    }
}
class MainFunction{
    static void Main(string[] args){
        //用多态来实现将移动硬盘U盘或者MP3插到电脑上进行读写数据

        //只需要改new的类型即可完成U盘、MP3等的转换
        MobileStorage ms = new USBFlashDisk();//new MP3();//new MobileHD();
```

```
            Computer cpt = new Computer(ms);
            cpt.CpuRead();
            cpt.CpuWrite();

            //下面是不在Computer里封装MobileStorage _ms时的实现方法
            // MobileStorage ms2 = new MP3();
            // Computer cpt2 = new Computer();
            // cpt.CpuRead(ms2);
            // cpt.CpuWrite(ms2);
            // ((MP3)ms2).PlayMusic();
        }
    }
```

补充：

1、虚方法和抽象类的使用差别

虚方法：当父类的方法有意义，知道该怎么实现时，且需要父类实例化时，就可以把父类定义成一个普通类，用虚方法来实现多态
抽象类：当父类的方法无意义，不知道该怎么实现时，就将父类定义成抽象类，把方法交给子类来实现，以此实现多态

2、抽象类中可以包含虚方法

当一个类中既需要一个抽象方法又需要一个虚方法时，可以把这个类定义为抽象类，这两个方法也都可以正常放进来