# 接口 （重难）

## 一、快速入门

- **现实世界的接口：**


。

- **编码世界的接口**

### 1.设计接口

```
package com.Interface;
/*这里定义的接口相当于现实中人为规定的
接口大小哇，线路，性能等等的规范，
这样才能让接口与设备之间吻合顺利插入后正常使用*/
public interface UsbInterface {
    //规定接口相关方法
    public void start();
    public void stop();
    //以后会规定更多的方法
}
```

### 2.设计设备Phone

```
package com.Interface;
//Phone类为了插上usbInterface接口 要规定/声明方法
//解读1.Phone类需要实现 UsbInterface接口
public class Phone implements UsbInterface{//手机实现接口
    @Override
    public void start() {
        System.out.println("手机开始工作...");
    }

    @Override
```

```
    public void stop() {
        System.out.println("手机停止工作...");
    }
}
```

## 2.设计设备Camera

```
package com.Interface;

//与手机同理
//实现接口 本质：接口方法(此时的接口方法是start()、stop()来自与UsbIntrface)的实现
public class Camera implements UsbInterface{//照相机实现接口
    @Override
    public void start() {
        System.out.println("相机开始工作...");
    }

    @Override
    public void stop() {
        System.out.println("相机停止工作...");
    }
}
```

## 3.设计主设备Computer

```
package com.Interface;

public class Computer {
    /*由于手机 和 相机都实现了接口功能
    * work入参调用接口对象相当于
    * Computer可以识别Phone 和 Camera了*/
    public void work(UsbInterface usbInterface){
        //接下来调方法
        usbInterface.start();
        usbInterface.stop();
    }
}
```

## 4.主函数Interface01

```java
package com.Interface;

public class InterfaceO1 {
    public static void main(String[] args) {
        //创建手机和相机
        Camera camera = new Camera();
        Phone phone = new Phone();
        //创建计算机
        Computer computer = new Computer();
        computer.work(phone);//把手机接入到计算机
        computer.work(camera);
    }
}
```

# 二、系统介绍接口

- **接口的语法**

```java
interface 接口名{
    //属性
    //方法(1.抽象方法2.默认实现方法3.静态方法)
}

//一个类要想(比如某种设备)要想实现接口，写法是
class 类名 implement 接口{//implement:实现
    //自己的属性
    //自己的方法
    //必须实现的接口的抽象等等
}
```

# 一、语法使用技巧

- **接口**

```java
package com.Interface.Inter02;
/*1.在jdk7之前 接口里的方法必须全是抽象方法不可以有实现了的方法
    在jdk8之后 接口里的方法可以有实现过后的方法了，
    但是需要default关键字修饰
  2. 在jdk8之后,可以实现静态方法*/
public interface AInterface {
    //属性
    public int n1 = 10;
    //方法
    //在接口中，抽象方法，可以省略abstract关键字
    public void hi();
    //在接口这实现一个方法
    default public void ok(){
        System.out.println("ok..");
    }
    //jdk8之后在接口这实现一个静态方法
    public  static void cry(){
        System.out.println("cry..");
    }
```

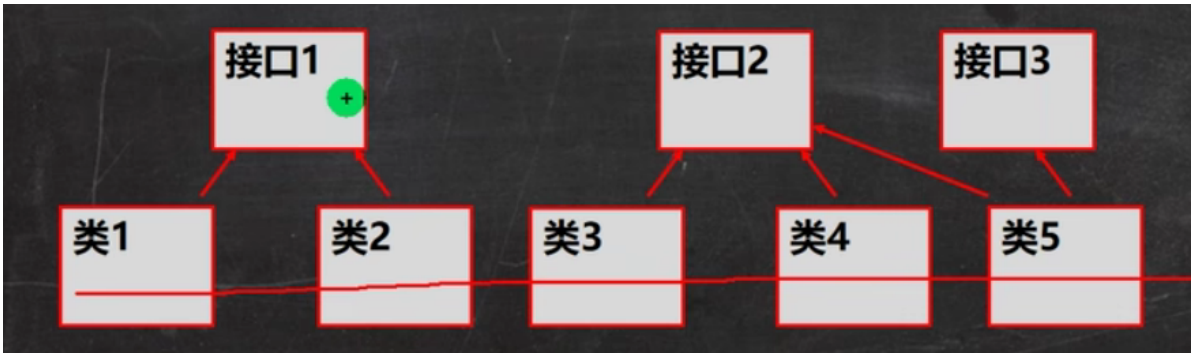
```
}
```

- **实现类**

```
package com.Interface.Inter02;

import com.Interface.Inter02.AInterface;

public class Interface02 {
    public static void main(String[] args) {

    }
}
/*1.如果一个类 implement实现 接口,
    就需要将该接口所有抽象类都实现
  */
class A implements AInterface {
    @Override
    //这是AInterface的抽象方法
    public void hi() {
        System.out.println("hi..");
    }
}
```

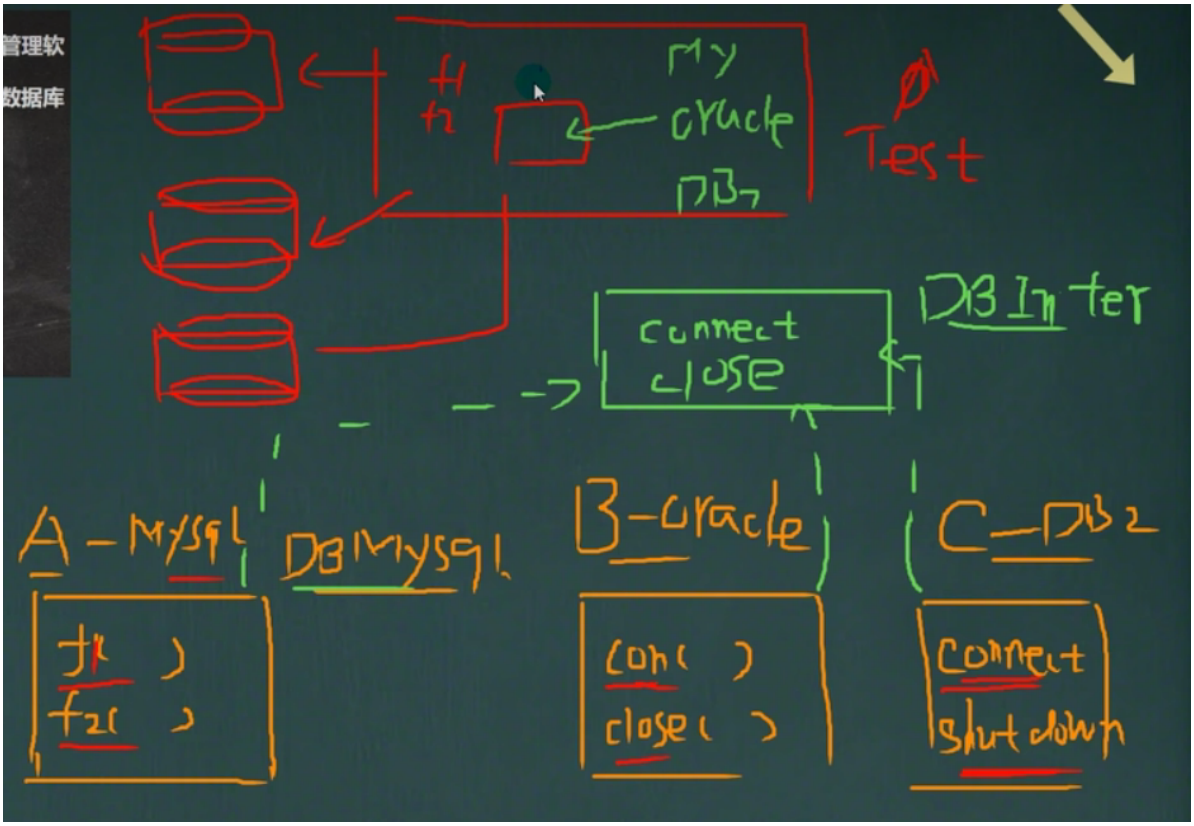
# 三、什么时候使用接口(难点)

- **使用接口的原因：为了控制和管理软件，项目经理可以定义一些接口，然后由程序员实现。若只单纯写类就 没有办法控制其 方法名、调用的形式等等,因此没有了接口就没有办法控制软件的子量了,最重要的是接口涉及到统一调用的问题.**




。

- **场景：3个程序员，编写三个类,分别完成对Mysql、Oracle、DB2的数据库连接Connection 和 Close**
- **关系图:**




。

# 一、体会接口编程的威力

## 1.接口DBInterface

```
package com.Interface.Inter03;
//组大佬定义好规范就可以回家睡觉觉了
public interface DBInterface {//接口这种东西项目经理写的
    public void connect();//连接方法
    public void close();//关闭方法
}
```

## 2.MysqlDB类

```
package com.Interface.Inter03;
//Mysql是组里面马仔1号写的
public class MysqlDB implements DBInterface {
    //马仔1号的任务是写Mysql相关功能的，必须按照老大的规范实现方法
    @Override
    public void connect() {//马仔一号不会乱取名字了，马仔2，3号都得按照这样规范写
        System.out.println("Mysql连接...");
    }
    @Override
    public void close() {
        System.out.println("Mysql关闭...");
    }
}
```

### 2.OracleDB类

```
package com.Interface.Inter03;
//马仔2号跟一号一样要按照老大的规范写connection和close操作了
public class OracleDB implements DBInterface{
    @Override
    public void connect() {
        System.out.println("Oracle连接...");
    }

    @Override
    public void close() {
        System.out.println("Oracle关闭...");
    }
}
```

### 3.Interface03主类 & 引入接口对象的类

```
package com.Interface.Inter03;

public class Interface03 {
    public static void main(String[] args) {
        //主函数中调用
        //当想要连接Mysql就new一个MysqlDB
        MysqlDB mysqlDB = new MysqlDB();
        //由于这里的t摆在了下面static了就不用创建对象了
        t(mysqlDB);//传mysqlDB这个对象给t，这就ok了
        //同理想要连接Oracle
        OracleDB oracleDB = new OracleDB();
        t(oracleDB);
    }
    /*理解:
    1.引入接口对象，由于各种类与接口建立关系，所以接口就包含了各种类和各种类的功能
    2.现在将这个接口对象安装到t里面,就像笔记本电脑边边开了个usb接口可以连接各种不同的设备
    */
    public static void t(DBInterface db){//DBInterface db接收其中一个对象
        db.connect();
        db.close();
    }
}
```

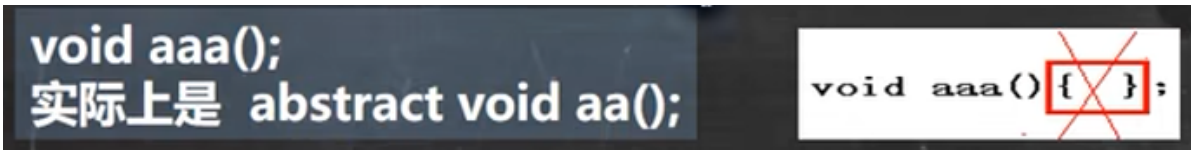
# 四、接口Detail

## Part 1：

1. **接口不能被实例化.与抽象类一样**

```
public class InterfaceDetail01{
    public static void main(String[] args){
        new IA();//会报错，因为接口不能被实例化
    }
}
interface IA{

}
```

2. **接口中所有方法是public方法,接口中抽象方法可以省略abstract关键字.**




。

3. **一个普通类实现接口,接必须将该接口的所有方法都实现.(将所有方法都实现Idea中快捷键：Alt+enter)。**

```
interface IA{
    void say();//修饰符 public、protected、private、默认都不能在接口上用
    void hi();
}
class Cat implement IA{
    void say(){//这样写会报错，因为实现接口的类中不会默认public修饰符，而是默认

    }
    public void hi(){//这才对

    }
}
```

4. **抽象类实现接口,可以不用实现接口的方法.**

```
abstract class Tiger implement IA{

}
```

- **分析：Tiger是抽象类啊，抽象类可以不实现方法的。所以实现Tiger接口的时候加上abstract关键字可以不实现接口的方法.**

## part 2:

5. **一个类可以同时实现多个接口.但继承不能**

```
interface IB{
    void hi();
}
interface IC{
    void say();
}
class AB implements IB,IC{

    @Override
    public void hi() {
        System.out.println("hi..");
    }

    @Override
    public void say() {
        System.out.println("say something..");
    }
}
```

6. **接口中的属性只能是final的,而且是public static final修饰符。例如**

```
//加入在接口中写了一个
int a = 1;
//实际上它的涵义是:
public static final int a = 1;//此处必须初始化
```

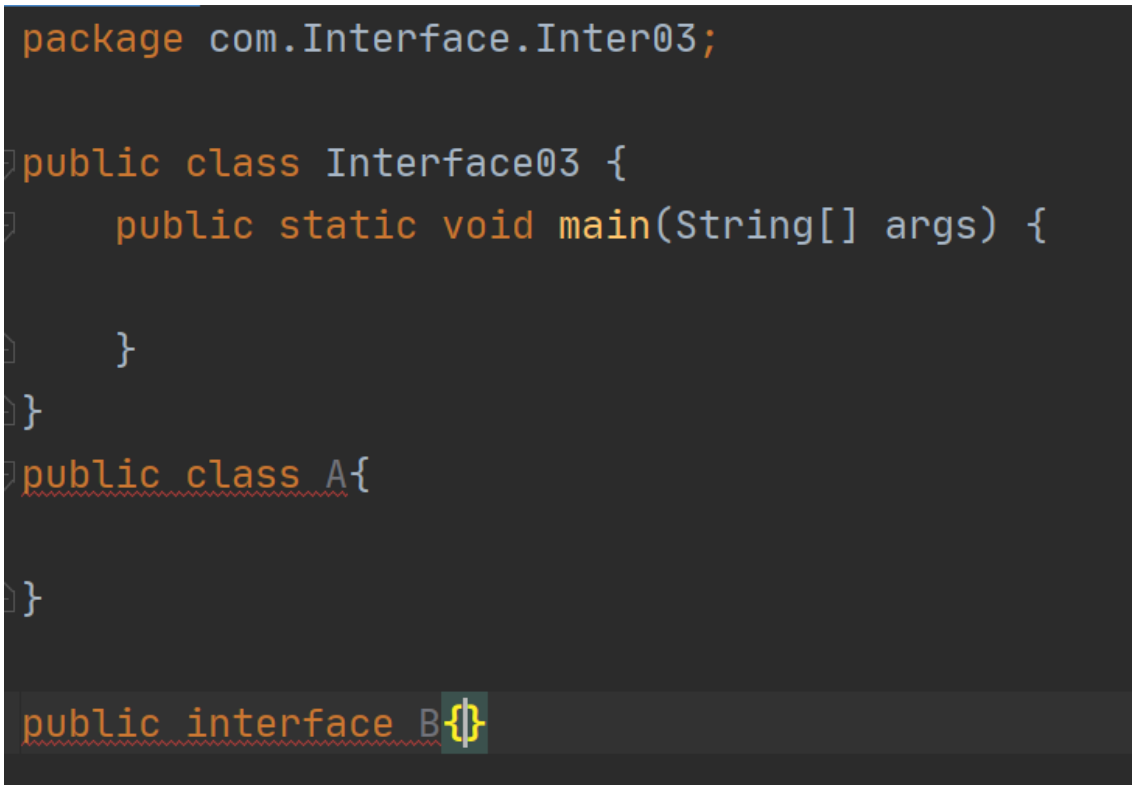
```
//证明 接口中的属性，是 public static final
public class InterfaceDetail01{//主函数
    public static void main(String[] args){
       //1.证明static
        System.out.println(IB.n1);
        //只因为静态类(即static)在主函数调用可以不new，直接(对象.啥啥啥),此处是(BI.n1)

        //2.证明final
        IB.n1 = 30;//若在这里在定义一次n1会出错,因为final在接口是最后一次的定义了
    }
}

interface IB{
    int n1 = 10;//3.证明public,在此处尝试用private,protected会报错，默认可以在实现接口类的属性上尝试,此处默认是public
    void hi();
}
```

7. **接口中属性的访问形式：接口名.属性名.**

8. **接口不能继承其它的类，但是可以继承多个别的接口.**

```
interface A{}
interface B{}
interface C extends A , B{}//C接口可以继承很多个接口，
//不要写成 interface C implement A,B{} 了
class D{}
interface C extends D{}//C是接口D是类，接口不能继承类，所以这样写是错滴
```

**9.接口的修饰符 只能是 public 和 默认,这点和类的修饰符是一样的.**

- 注释：一个文件不能有两个public 类
- 

```
package com.Interface.Inter03;

public class Interface03 {
    public static void main(String[] args) {

    }
}
public class A{

}

public interface B{}
```

。

- **正确做法：一个文件一个public类（interface可以理解为特殊的类）**

```
package com.Interface.Inter03;
//组大佬定义好规范就可以回家睡觉觉了
public interface DBInterface {//接口这种东西项目经理写的
    public void connect();//连接方法
    public void close();//关闭方法
}
```

。

## 五、练习1：

### 1.imlpement的影响

```
package com.Interface.Excer01;

public class InterfaceExcer01 {
    public static void main(String[] args) {
        B b = new B();
        //B对象实现了A自然可以直接调用其属性
        System.out.println(b.a);//23
        //由于static所以A可以直接调用
        System.out.println(A.a);//23
        //同理因为B实现了A,有因为A的属性有static功能，B调用A的属性自然也有static功能
        System.out.println(B.a);//23
    }
}
interface A{
    int a = 23;//等价于 public static final  int a =23;
}
class B implements A{

}
```

## 六、接口和继承类的比较

- **场景：猴子天生会爬树，但是想跟鱼学游泳.（精华内容）**

### 1.父类 Monkey

```
package com.Interface.Extends_VS_Interface;

public class Monkey {//猴子父亲(父类)
        private String name;
        public Monkey(String name){
            setName(name);
        }

        public void climbing(){
            System.out.println(name + "天生会爬树");
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }
}
```

## 2.接口 Fish

```
package com.Interface.Extends_VS_Interface;

public interface Fish {//鱼与猴子没有血缘关系,谈不上继承关系啦
    //鱼天生会游泳
    public void Swimming();
}
```

## 3.子类 LittleMonkey

```
package com.Interface.Extends_VS_Interface;
//小猴子继承了老猴子，虽然继承不了鱼的游泳技能但是跟鱼学了游泳的技能
public class LittleMonkey extends Monkey implements Fish{
    public LittleMonkey(String name){//重写父类构造器
        super(name);
    }
    //子类因为继承了爬树技能所有不用在这里写多一次climbing技能,除非你技能进化了(有了自己特殊技能)可以重写父类climbing方法添加新的天生技能
    public void Swimming(){//学习技能
        System.out.println(getName()+"跟鱼学会了游泳");
    }
}
```

## 4.主方法Extends_VS_Interface

```
package com.Interface.Extends_VS_Interface;

public class Extends_VS_Interface {
    public static void main(String[] args) {
        LittleMonkey PoHou = new LittleMonkey("泼猴");
        PoHou.climbing();
        PoHou.Swimming();
    }
}
```

- **总结：子类继承了父类可以直接继承父类的所有功能，但是子类想要扩展功能，就需要通过 接口实现(也就是后天学习)来获得技能.**
- **继承价值在于：解决代码的复用性 和 可维护性.**
- **接口价值在于：设计好各种规范(方法),让其它类去实现这些方法,更加灵活、规范.'**

- **接口在一定程度上可以实现解耦（即接口具有 规范性 和 动态绑定机制）**

# 七、接口的多态(特性)

## 一、接口多态

### 1. 接口体现的多态

```
package com.Interface.InterfacePloy;

public class InterfacePolyO1 {
    public static void main(String[] args) {
        //接口多态的体现
        /*1.解读：IF是接口类型，ifO1是其变量 Monkey 和 Fish 是实现了IF接口的对象
          2.接口类型的变量 ifO1 可以指向 实现了IF接口的对象*/
        IF ifO1 = new Monkey();
        ifO1 = new Fish();


    }
}
interface IF{//某种接口

}
class Monkey implements IF{}
class Fish implements IF{}
```

### 2. 继承体现的多态

```
package com.Interface.InterfacePloy;

public class InterfacePolyO1 {
    public static void main(String[] args) {
        //继承体现的多态
        /*父类类型的变量 a 可以指向 继承了AA的子类对象*/
        AA a = new BB();//向上转型
        a = new CC();

        /*//向下转型：例如AA强转成了BB
        AA aa = new AA();
        BB bb = (BB) aa;*/
    }
}
//继承体现的多态
class AA{}
class BB extends AA{}
class CC extends AA{}
```

## 二、接口多态数组

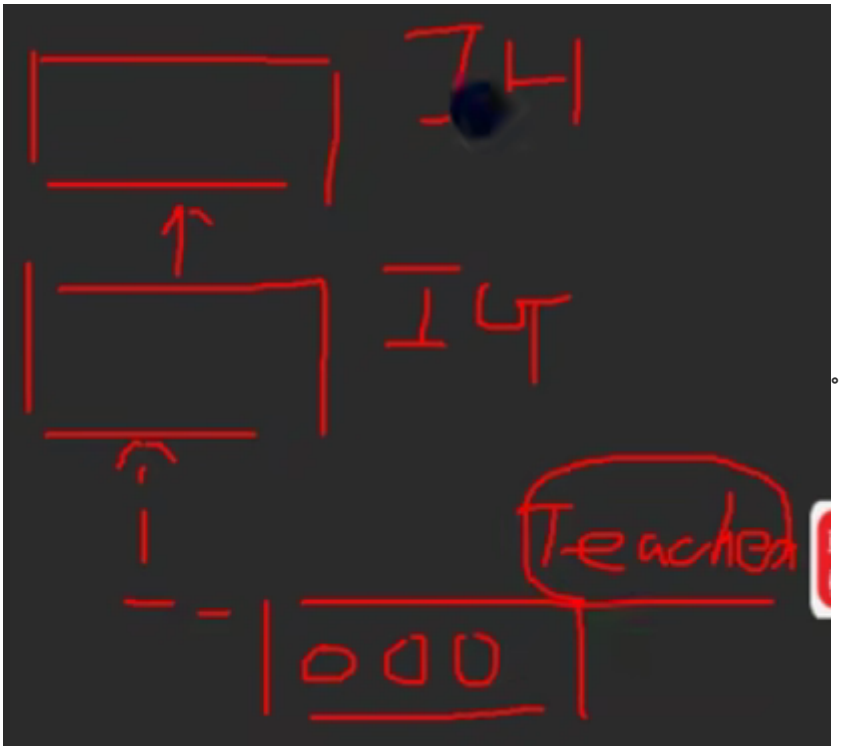
```
package com.Interface.InterfacePolyArr;

public class InterPloyArrO1 {
    public static void main(String[] args) {
```

```
        //接口有数组,数组可实例化(即new),但单纯的接口不能被实例化与抽象类一样
        Usb[] usbs = new Usb[2];
        usbs[0] = new Phone();
        usbs[1] = new Camera();

        //遍历数组
        for (int i = 0; i < usbs.length ; i++){
            usbs[i].work();//动态绑定机制 ,因为统一都有work()
            //和前面一样,我们任然需要进行类的向下转型
            /*解释instance：判断数组的运行类型是不是Phone*/
            if(usbs[i] instanceof Phone){
                //向下转型
                ((Phone) usbs[i]).call();//解释：将usbs[i]强转成 Phone,再‘ . ’调用专
属于Phone的call方法
            }
        }
    }
}

interface Usb{
    void work();
}
interface Rsb{}
class Phone implements Usb,Rsb{//一个类可以实现多个接口，一个接口可以继承多个接口，但不可
以继承类
    public void call() {
        System.out.println("手机正在接听电话...");
    }

    @Override
    public void work() {
        System.out.println("手机正在工作...");
    }
}
class Camera implements Usb{
    @Override
    public void work() {
        System.out.println("相机正在工作...");
    }
}
```

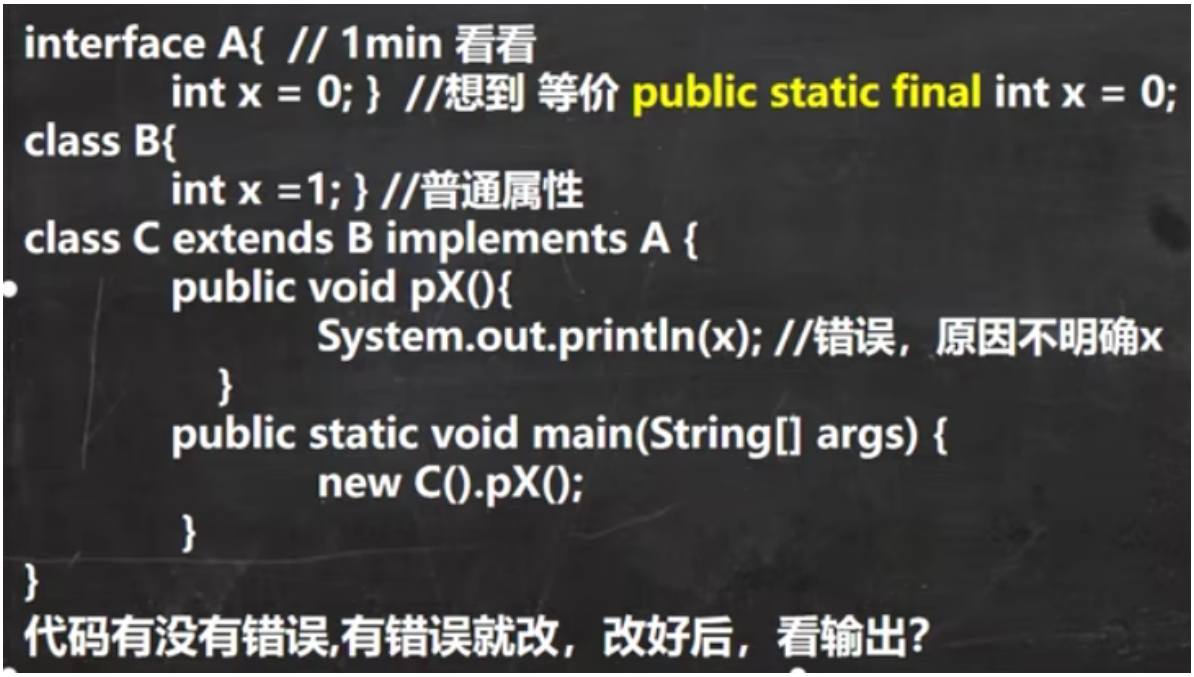
## 三、 接口传递现象

```
package com.Interface.InterfacePolyPass;
/*演示多接口态传递现象*/
public class InterfacePolyPassO1 {
    public static void main(String[] args) {
        /*1.IG接口 可以指向 Teacher类
        * 因为teacher类实现了IG接口,
        * 他俩有师徒关系 */
        IG ig = new Teacher();
        /*2.IH 不可以指向 Teacher类,他俩没有师徒关系*/
       /* IH ih = new Teacher();//错误写法*/

        /*3.这是IG继承了IH 而Teacher实现了IG接口,
            实际上相当于Teacher也实现了IH这个接口*/
```

```
        //这就是多态传递现象
        IH ih = new Teacher();
    }
}

interface IH{
    void ih();
}
interface IG extends IH{}//3.若IG 继承了 IH 那Teacher就与IH有关系啦！祖师爷啊(师傅的老爸叫什莫)！
class Teacher implements IG{
    //因为IG继承了IH，所以Teacher类要跟师公学技能
    //重写师公方法
    @Override
    public void ih() {
        System.out.println("师公技能...");
    }
}
```




# 八、练习




。

- x若是找父类可以super(x) 或者 super.x.
- x若是找接口A可以写A.x.