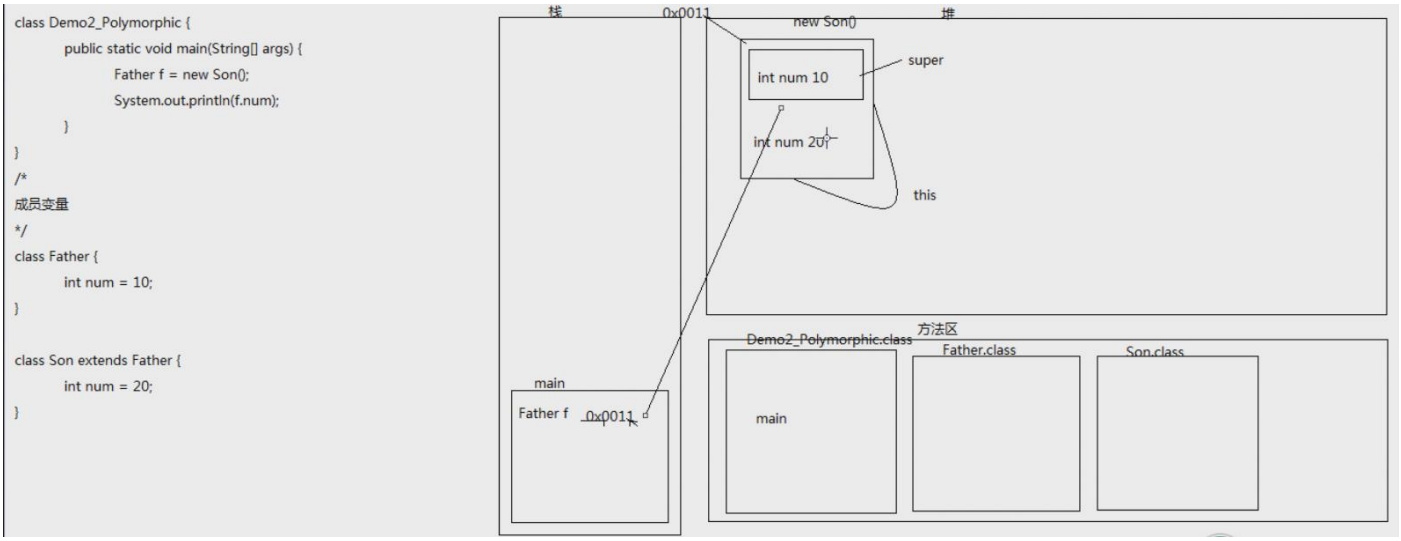
●09.01_面向对象(多态的概述及其代码体现)

- A:多态(polymorphic)概述
    - 事物存在的多种形态
- B:多态前提
    - a:要有继承关系。
    - b:要有方法重写。
    - c:要有父类引用指向子类对象。
- C:案例演示
    - 代码体现多态

```
public class Demo9 {
    public static void main(String[] args) {
        Animal c = new Cat();//父类引用指向子类的对象叫多态
      c.eat();//Cannot resolve method 'eat' in 'Animal'注释掉父类中的eat 方法之后,发现编译通不过,
        /*
        结论:在多态中编译的时候会检查父类中是否有被重写的eat 方法
          编译看右边(父类)
             运行看左边(子类)
         */
    }
}
class Animal{
    String color="黑色";
    public void eat(){
        System.out.println("动物在吃东西,"+color);
    }
}
//继承
class Cat extends Animal{
    String color = "白色";
    public void eat(){
        System.out.println("猫吃鱼,"+color);
    }
}
```

●09.02_面向对象(多态中的成员访问特点之成员变量)

```java
class Demo2_Polymorphic {
    public static void main(String[] args) {
        Father f = new Son();                    //父类引用指向子类对象
        System.out.println(f.num);
    }
}
/*
成员变量
*/
class Father {
    int num = 10;
}

class Son extends Father {
    int num = 20;
}
```



- 成员变量
  - (编译看父类，运行看父类)(和普通的继承一样)

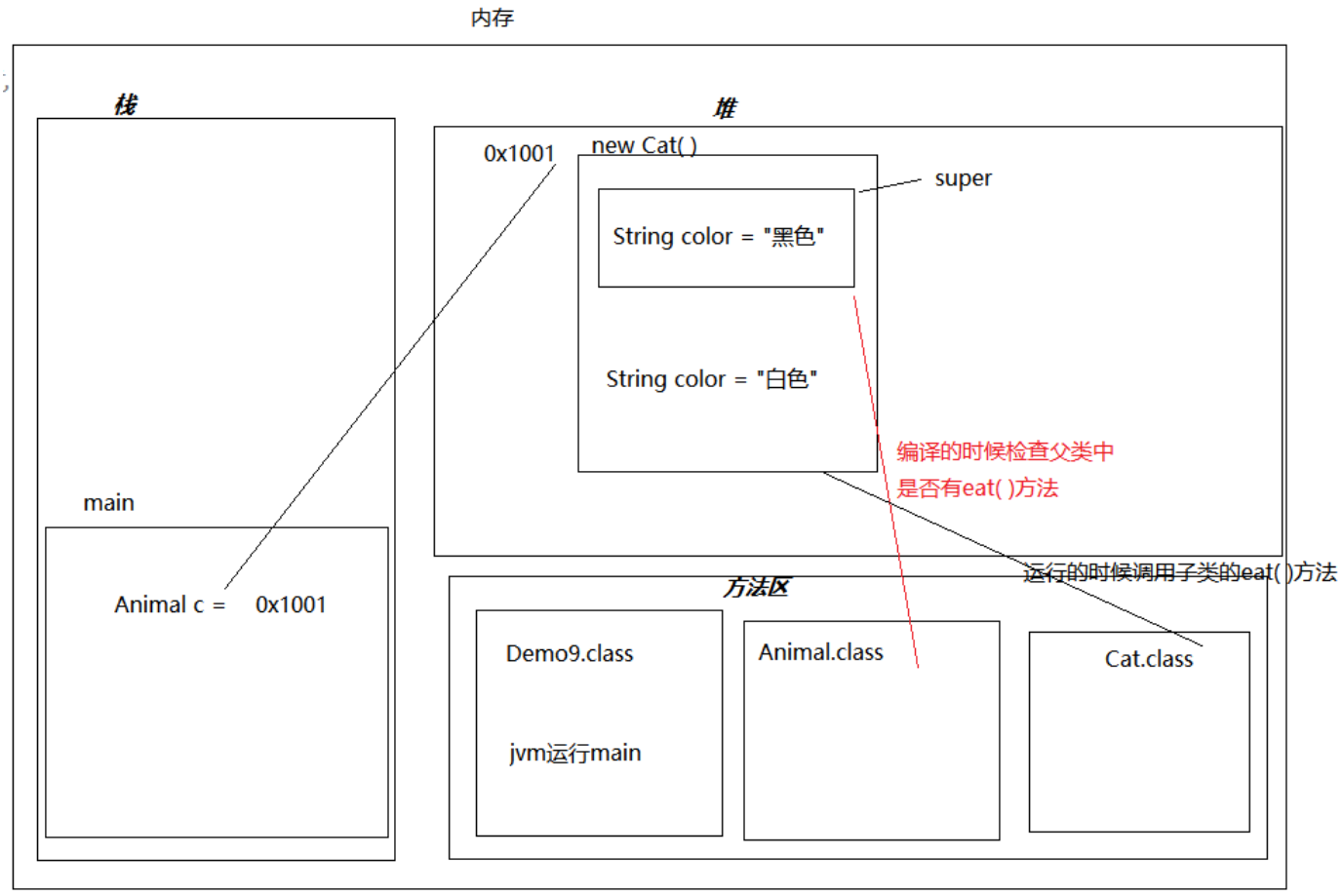
●09.03_面向对象(多态中的成员访问特点之成员方法)

```
成员方法
编译看左边(父类)，运行看右边(子类)。动态绑定
*/
class Father {
    int num = 10;
    public void print() {
        System.out.println("father");
    }
}

class Son extends Father {
    int num = 20;

    public void print() {
        System.out.println("son");
    }
}
```

```
Father f = new Son();
//f.print();
```

- 成员方法
    - 编译看左边(父类)，运行看右边(子类)。

出现动态绑定



多态有动态绑定的现象

●09.04_面向对象(多态中的成员访问特点之静态成员方法)

```
class Father {
    int num = 10;
    public void print() {
        System.out.println("father");
    }

    public static void method() {
        System.out.println("father static method");
    }
}

class Son extends Father {
    int num = 20;

    public void print() {
        System.out.println("son");
    }

    public static void method() {
        System.out.println("father static method");
    }
}
```

```
Father f = new Son();
//f.print();
f.method();
```

执行父类的

相当于 father 的 method 方法

静态方法

- 编译看左边(父类)，运行看左边(父类)。
- (静态和类相关，算不上重写，所以，访问还是左边的)，当普通类里的情况来理解
- 只有非静态的成员方法,编译看左边,运行看右边

●09.05_面向对象(钢铁侠的故事)

- A:案例分析
  - 多态的现象

```
public class Demo10 {
    public static void main(String[] args) {
        Person p = new IronMan();
        p.business();
        p.saveWorld();
```

```
//        p.paoNiu();
        //我们发现了一个问题,在多态中,父类的引用不能调到子类自己特有的方法,怎么解决这个问题?多态中的短板
    }
}
class Person{
    String name = "托尼";
    public void business(){
        System.out.println(name+"卖军火");
    }
    public void saveWorld(){
        System.out.println("有一颗正义的心");
    }
}
class IronMan extends Person{
    String name = "钢铁侠";
    //子类自己特有的方法
    public void fly(){
        System.out.println("可以飞");
    }
    //子类自己特有的方法
    public void fire(){
        System.out.println("发射火箭炮");
    }
    //重写
    public void saveWorld(){
        System.out.println(name+"拯救受灾难的人群");
        System.out.println("攻击坏人");
        fly();
        fire();
    }
    //子类自己特有的方法
    public void paoNiu(){
        System.out.println("经常晚上抱着自己的秘书飞,带她展示 IronMan 盔甲的功能");
    }
}
```

●09.06_面向对象(多态中向上转型和向下转型)

- A:案例演示
    - 详细讲解多态中向上转型和向下转型

```
public class Demo10 {
    public static void main(String[] args) {
        Person p = new IronMan();  //子类的数据类型被提升为父类的数据类型
```

```
        p.business();
          p.saveWorld();

//          p.paoNiu();
          //我们发现了一个问题,在多态中,父类的引用不能调到子类自己特有的方法,怎么解决这个问题?多态中的短板
        System.out.println("---------------");

          //向下强制转型
        IronMan ir = (IronMan) p;
          //向下强制转型之后就可以直接调用子类特有的方法
        ir.paoNiu();
          ir.fire();
          ir.fly();
    }
}
class Person{
    String name = "托尼";
    public void business(){
        System.out.println(name+"卖军火");
    }
    public void saveWorld(){
        System.out.println("有一颗正义的心");
    }
}
class IronMan extends Person{
    String name = "钢铁侠";
    //子类自己特有的方法
   public void fly(){
        System.out.println("可以飞");
    }
    //子类自己特有的方法
   public void fire(){
        System.out.println("发射火箭炮");
    }
    //重写
   public void saveWorld(){
        System.out.println(name+"拯救受灾难的人群");
        System.out.println("攻击坏人");
        fly();
        fire();
    }
    //子类自己特有的方法
   public void paoNiu(){
        System.out.println("经常晚上抱着自己的秘书飞,带她展示 IronMan 盔甲的功能");
    }
}
```

多态的应用场景之一：

```
public class Demo1 {
    public static void main(String[] args) {
        Oringe o = new Oringe();
        o.name = "橘子";
        o.juicer();

        Banana b = new Banana();
        b.name = "香蕉";
        b.juicer();

        Apple a = new Apple();
        a.name = "苹果";
        a.juicer();
    }
}
class Fruit{
    String name;

    public void juicer(){
        System.out.println("榨汁成"+name+"汁");
    }
}
class Oringe extends Fruit{

}
class Banana extends Fruit{

}
class Apple extends Fruit{

}
```

```
public class Demo1 {
    public static void main(String[] args) {
        Fruit f= new Fruit();
        Oringe o = new Oringe();
        o.name = "橘子";
        f.juicer(o); // Fruit f =new Oringe();

        Banana b = new Banana();
        b.name= "香蕉";
        f.juicer(b); //Fruit f = new Banana();

        Apple a = new Apple();
        a.name ="苹果";
        f.juicer(a);
```

```
    }
}
class Fruit{
    String name;

    public void juicer(Fruit f){    //简单工厂设计模式
      System.out.println("榨汁成"+f.name+"汁");
    }
}
class Oringe extends Fruit{

}
class Banana extends Fruit{

}
class Apple extends Fruit{

}
```

●09.07_面向对象(多态的好处和弊端)

- A:多态的好处
    - a:提高了代码的维护性(继承保证)
    - b:提高了代码的扩展性(由多态保证)
- B:案例演示
    - 多态的好处
    - 可以当作形式参数,可以接收任意子类对象
- C:多态的弊端
    - 不能使用子类的特有属性和行为（强转后才可以使用子类特有的属性和方法）。
- D:案例演示

●09.08_面向对象(多态中的题目分析题)

- A:看下面程序是否有问题，如果没有，说出结果

```
class Fu {
```

```
    public void show() {
        System.out.println("fu--show");
    }
}
class Zi extends Fu {
    public void show() {
        System.out.println("zi__show");
    }
    public void method() {
            System.out.println("zi__method");
        }
    }
public class Test1Demo {
    public static void main(String[] args) {
        Fu f = new Zi();
        f.method();//会报错,父类引用不能调用子类特有的方法
        f.show();   //输出 zi____show
    }
}
```

- B:看下面程序是否有问题，如果没有，说出结果

```
class A {
    public void show() {
        show2();
    }
    public void show2() {
        System.out.println("我");
    }
}
class B extends A {
    public void show2() {
        System.out.println("爱");
    }
}
class C extends B {
    public void show() {
       super.show();
    }
    public void show2() {
        System.out.println("你");
    }
}
public class Test2DuoTai {
    public static void main(String[] args) {
        A a = new B();
        a.show();
```

```
        B b = new C();
        b.show();
    }
}
```

```
爱
你
```

课堂练习：

写一个汽车类
属性：品牌
用多态的方式
汽车输品牌，及价格
家用车
出租车
卡车

```
public class Demo2 {
    public static void main(String[] args) {
        Car c = new Car();
        Taxi t = new Taxi();
        t.brand = "红旗";
        t.price = 1000000;
        c.doSome(t); //Car c= new Taxi()

        HomeCar hc = new HomeCar();
        hc.brand = "大众";
        hc.price = 120000;
        c.doSome(hc); //Car c = new HomeCar();

        Truck tr = new Truck();
        tr.brand = "解放牌";
        tr.price = 5000000;
        c.doSome(tr);//Car c = new Truck();

    }
}
/*
写一个汽车类
属性：品牌
用多态的方式
输出汽车品牌，及价格
家用车
出租车
卡车
 */
class Car{
    String brand;//品牌
   int price;//价格
```

```
    public void show(){
        System.out.println(brand+"--"+price);
    }

    public void doSome(Car c){
        //调用继承父类的方法
       c.show();
        //先根据实际类型向下转型,再调用子类自己特有的方法
       if(c instanceof HomeCar){  //instanceof 表示判断是否属于哪个数据类型(用来测试对象 c 是否为一个类 HomeCar 的实例对象)
            HomeCar hc = (HomeCar) c;//向下转型
          hc.zaiKe();
        }else if(c instanceof Taxi){
            Taxi t = (Taxi)c;
            t.yunYin();
        }else if(c instanceof Truck){
            Truck tr = (Truck)c;
            tr.carryProduct();
        }

    }
}
class HomeCar extends Car{
    //载客
    public void zaiKe(){
        System.out.println("我可以载客");
    }
}
class Taxi extends Car{
    //运营商用
    public void yunYin(){
        System.out.println("收费载客");
    }
}
class Truck extends Car{
    //载货
    public void carryProduct(){
        System.out.println("卡车载货");
    }
}
```

●09.09_面向对象(抽象类的概述及其特点)

- A:抽象类概述

- 抽象就是看不懂的

- B:抽象类特点
  - a:抽象类和抽象方法必须用 abstract 关键字修饰
    - abstract class 类名 {}
    - public abstract void eat();//抽象方法没有实现，就是没有{ }
  - b:抽象类不一定有抽象方法，有抽象方法的类一定 是抽象类或者是接口
  - c:抽象类不能实例化那么，抽象类如何实例化呢?
    - 按照多态的方式，由具体的子类实例化。其实这也是多态的一种，抽象类多态。
  - d:抽象类的子类
    - 要么是抽象类
    - 要么重写抽象类中的所有抽象方法

- C:案例演示

```
public class Demo4 {
    public static void main(String[] args) {
        Worker w = new ZhangSan();  //抽象类如果要创建对象,得用多态,用子类创建对象
       w.show();
        ZhangSan zs = (ZhangSan) w;
        zs.work();
    }
}

abstract class Worker{
    public abstract void work();//抽象方法不要以后{},也就是,不用写方法的实现部分(方法体)

    //抽象类中可以有普通的成员方法
   public void show(){
        System.out.println("我是 show 方法");
    }
}
class ZhangSan extends Worker{
    @Override
    public void work() {    //发现继承了抽象类Worker 的子类被强制重写抽象类中的抽象方法
      System.out.println("我要工作,赚钱取老婆");
    }
```

```
}
/*
     抽象类提供了一个标准,强制子类来重写抽象方法,抽象类中可以没有抽象方法,但是没有意义,与抽象类存在的初衷有冲突
     可以有普通的方法,但是,一般开发中抽象类只写抽象方法,不写普通的成员方法

 */
```

JDK14 版本的新特性

接口中可以有抽象方法

default 修饰的方法

static 修饰的方法

实现了接口的子类,被强制要求重写接口中的抽象方法

```
package com.sxt.test;
public class Demo1 {
    public static void main(String[] args) {
        B b = new B();
        b.show();
        b.display();
        System.out.println(A.getMsg());
    }

}
/*
JDK14 版本的新特性
接口中可以有抽象方法
default 修饰的方法
static 修饰的方法
实现了接口的子类,被强制要求重写接口中的抽象方法
 */
interface A{
    public void show();

    default void display(){
        System.out.println("display 方法");
    }
    public static String getMsg(){
        return "getMSG 方法";
    }
}
class B implements A{

    @Override
```

```
    public void show() {
        System.out.println("show");
    }
}
```

●09.10_面向对象(抽象类的成员特点)

- A:抽象类的成员特点
    - a:成员变量：既可以是变量，也可以是常量。abstract 是否可以修饰成员变量?

不能修饰成员变量,只能修饰类或方法

    - b:构造方法：有。
        - 用于子类访问父类数据的初始化。
    - c:成员方法：既可以是抽象的，也可以是非抽象的。(一般情况下都是抽象的)
- B:案例演示
    - 抽象类的成员特点
- C:抽象类的成员方法特性：
    - a:抽象方法 强制要求子类做的事情。
    - b:非抽象方法 子类继承的事情，提高代码复用性。

●09.11_面向对象(宦官制度)

- 案例演示宦官制度 Eunuch : 行为 spay
    - 抽象类的作用

```
public class Demo5 {
    public static void main(String[] args) {
        ZhenHe zh = new ZhenHe();
        zh.spay();
        zh.xiaXiYang();
    }
}
//宦官的类
abstract class Eunuch{
    public abstract void spay();//阉割
```

```
}
class ZhenHe extends Eunuch{
    @Override
    public void spay() {
        System.out.println("用牙签,用石头,用刀切");
    }
    public void xiaXiYang(){
        System.out.println("下西洋,发现印度板块,促进了明朝与印度大洋洲的贸易交流");
    }
}
```

洗衣服

洗涤要求:干洗

实现抽象类,并写一个子类创建实例

```
public class Demo6 {
    public static void main(String[] args) {
        Dao d = new Dao();
        d.ganXi();
    }
}

abstract class Clother{
    public abstract void ganXi();
}
class Dao extends Clother{
    @Override
    public void ganXi() {
        System.out.println("干洗,水洗不负责");
    }
}
```

●09.15_面向对象(抽象类中的面试题)

- A:面试题 1
    - 一个抽象类如果没有抽象方法，可不可以定义为抽象类? 如果可以，有什么意义?

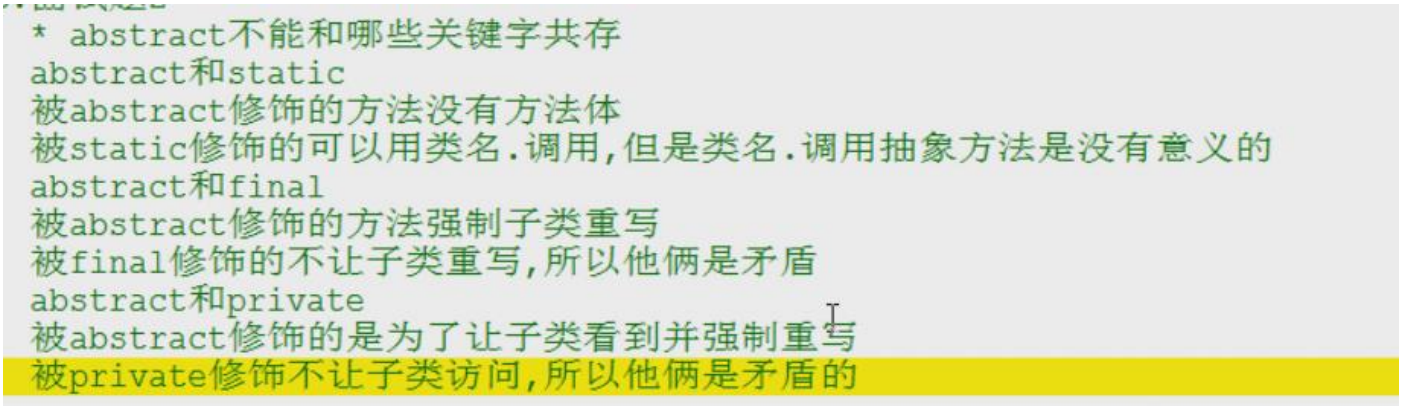
可以,没有什么意义,它唯一目的是:这么做目的只有一个,就是不让其他类创建本类对象,交给子类用多态的形式完成创建对象

- B:面试题 2

- abstract 不能和哪些关键字共存，说出为什么

static、final、private 来搭配 **abstract**

static: 不能,static 修饰的方法可以用类名.方法名调用,说明这个类是可以创建本类对象的,而 abstract 不能创建本类对象,所以不能搭配 final: 不能,final 修饰的类不能被继承,修饰的方法不能被重写,abstract 修饰的类,要求要被子类继承使用 private: 不能,private 修饰的方法不能被继承和重写,与 abstract 冲突



●09.16_面向对象(接口的概述及其特点)

- A:接口概述
  - 从狭义的角度讲就是指 java 中的 interface
  - 从广义的角度讲对外提供规则的都是接口

USB 接口,HDML 视频,TYPE_C,VGA 台式机连接显示器视频接口

  - 接口内的方法全部为抽象方法
- B:接口特点
  - a:接口用关键字 interface 表示
    - interface 接口名 {}
  - b:类实现接口用 implements 表示
    - class 类名 implements 接口名 {}

```
interface Inter {
    public abstract void print();                    //接口中的方法都是抽象的
}

class Demo implements Inter {
    public void print() {
        System.out.println("print");
    }
}
```

- c:接口不能实例化
  - 那么，接口如何实例化呢?
  - 按照多态的方式来实例化。

```
public static void main(String[] args) {
    //Inter i = new Inter();          //接口不能被实例化,因为调用抽象方法没有意义
    Inter i = new Demo();
    i.print();
}
```

- d:接口的子类
  - a:可以是抽象类。但是意义不大。

```
interface Inter {
    public abstract void print();                    //接
}

abstract class Demo implements Inter {
    /*public void print() {
        System.out.println("print");
    }*/
}
```

  - b:可以是具体类。要重写接口中的所有抽象方法。(推荐方案)

定义普通类　重写抽象方法

- C:案例演示
  - 接口特点(Phone)

```
public class Demo1 {
    public static void main(String[] args) {
       int num =  Phone.a;
        System.out.println(num);
        Phone p = new HuaWei();//接口不能创建实例,只能通过多态的方法让子类创建实例
       p.call();
        p.sendMessage();
```

```
        HuaWei hw = (HuaWei) p;
        hw.kuaPing();
    }
}
interface Phone{
    public void call();  //在接口中定义的所有方法,默认就是抽象的方法,会隐式自动加上 abstract
    public void sendMessage();
    int a = 10;//接口里面定义的所有变量都成了常量,底层会自动加上 public static final,一般接口中比较少申明变量
}
class HuaWei implements Phone{

    @Override
    public void call() {
        System.out.println("必须可以打电话");
    }

    @Override
    public void sendMessage() {
        System.out.println("必须要能发短信");
    }
    public void kuaPing(){
        System.out.println("跨屏显示");
    }
}
```