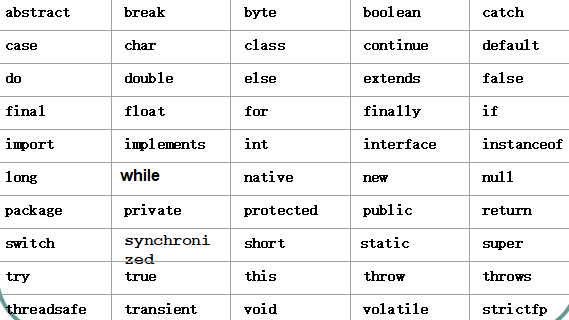
# JavaSE考试总结

## 一、补充

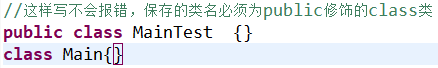
1.Java中的关键字

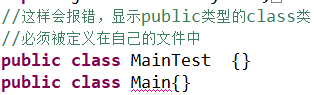
注意**instanceof**关键字的o是小写的，因此instanceOf是可以作为标识符的。



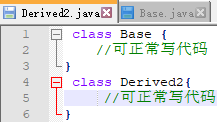
2.对以.java格式的文件进行编译就是在命令提示符窗口中进行编译。这里要注意一个问题，就是该java文件保存时，类名必须和保存的文件名完全一致，然后再.java。使用javac命令将该java文件编译称字节码文件即.class文件，然后再使用java命令运行该文件。

3.对一个java类型的文件进行编译，注意，**每一个public类型的class类保存时必须对应一个.java文件**。即不能在记事本中写两个public类然后以.java的格式保存，否则在编译的时候会报错。但是**在一个public修饰的class类下面还可以接着写其他权限的class类而不会报错，最后保存的时候以public修饰的class的类名保存**。

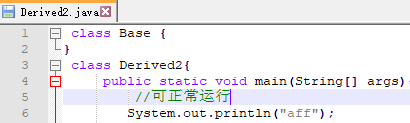




当class类都没有被public修饰时，保存时类名可以是该文件中的随便一个class类的类名。下面代码可以命名为Derived2.java也可以命名为Base.java，编译的时候都不会报错。这种在编译时会生成两个对应的.class文件。在命令提示符窗口运行时会报错，因为都没写main方法。只要有一个写了main方法，且保存的是对应的类名，则可以正常运行。



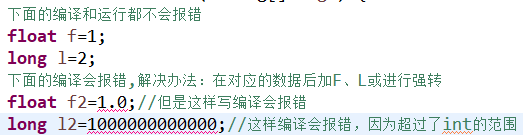
下面的则可以正常运行：



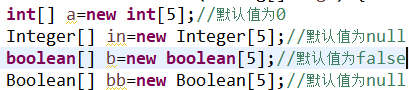
4.原生数据类型：八种基本数据类型。

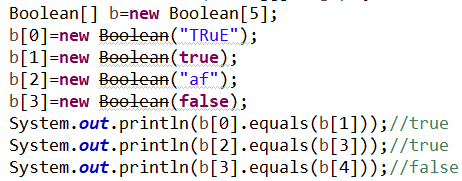
5.**java不是纯的面向对象的语言，不纯的地方在于其基本数据类型不是对象**。

6.在基本数据类型中，整型默认为int类型，浮点型默认为double类型。



8.**注意八种基本数据类型的数组，其默认值为该基本数据类型对应的默认值**。而这八种基本数据类型对应的**包装类所形成的数组，其默认值则为null**。注意八种基本数据类型对应的包装类都被final，即都不可以被继承，String也被final修饰也不能被继承。



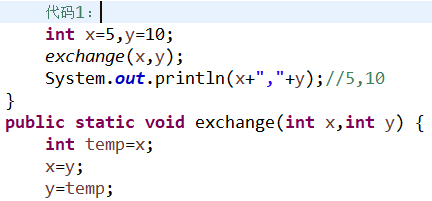


报错

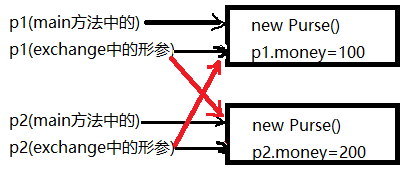
9.注意用equals()方法时，该方法前面不能为null类型的值，否则在运行时会报错，报空指针异常。equals()括号内可以为null。使用equals()方法有个小技巧，就是确定的不为null的值放前面，不确定的放括号内，这样就不会报空指针异常。注意除null外(因为不允许前面为null)，任何和equals(null)比相等的结果都为false。

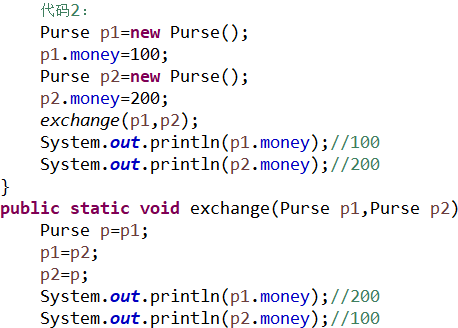
10.特别注意**在继承中，子类必须调用父类的构造器，默认情况下子类会自动调用父类的无参构造器，若父类没有无参构造器则必须自己手动用super()方法在子类构造器中调用父类的构造器，否则系统会报错**。

11.下面代码1中的x和y调用exchange(x,y)方法之后的值并没有被交换。exchange(int x,int y)中的形参x,y只是与传入的实参x,y在样式上是一样的，exchange()方法中的形参可以任意表示，如用exchange(int a,int b)这样表示就好理解了，只是将实参x,y的值对应赋值给a和b，并不会改变本身的大小。

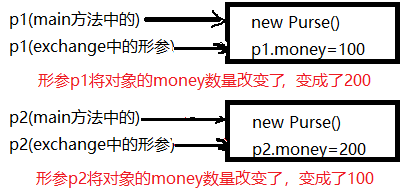


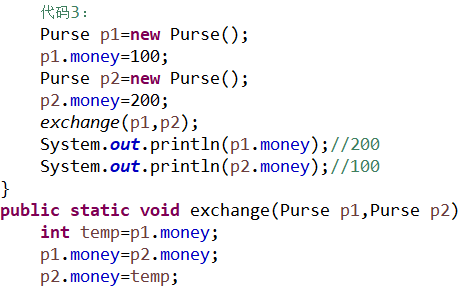
代码2可借助下面的图来分析，在调用exchange(p1,p2)方法时，将main方法中Purse的引用p1,p2的地址一起作为实参分别赋值给exchange()方法中的形参p1,p2。此时实参p1与形参p1的地址值指向的都是同一个Purse对象，实参p2与形参p2的地址值指向的也是同一个Purse对象。当在exchange()方法中将形参p1指向的地址值和p2指向的地址值进行交换，形参p1指向的是实参中p2的地址(它两则指向的是相同的对象)，而形参p2则指向的是实参p1的地址(它两指向的则是相同的对象)。形参p1指向的地址改变了并不会影响实参p1本身所指向的地址，同理p2也是一样的。



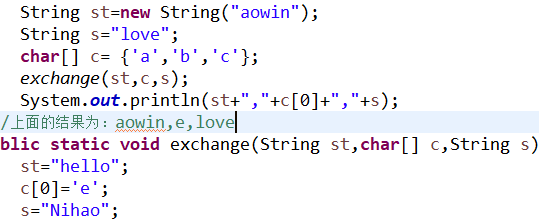


将上述代码该成下面的代码3，则结果就不一样了。相当于实参p1与形参p1指向相同的对象，但是由于形参p1将对象的内容改变了，将Purse的money数量变成了200，所以实参p1指向的对象money也改变了。p2同理。





注意String类型比较特殊，因为String类型即使是new出来的其对象引用打印出的也不是地址，而是字符串类型。应该把它当成基本数据类型中参数的传递。

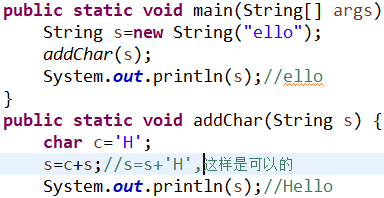


结论：通过上述的分析可知，**引用类型参数的传递过程，调用方的传入的实参变量和接收方的形参变量，指向的是同一个对象。双方任意一方对这个对象的修改，都会影响对方（因为指向同一个对象嘛）；基本类型参数的传递过程，是调用方值的复制。双方各自的后续修改，互不影响**。

12.构造函数是不能被覆写(重写)的，但是可以被重载。

13.任何抽象方法都不能被private、final、static修饰，因为抽象方法是要重写的。

14

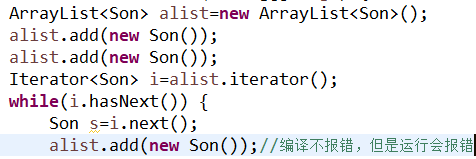


15.java中既能修饰变量又能修饰接口的方法的关键字有public、static。接口中的方法默认是public abstract修饰的，无方法体。但是还可以有default或者static修饰的方法，必须有方法体。

16.在类中，只要看到该有有参构造器就要考虑是否写了无参构造器，若没有则创建对象则不能用无参的方式创建。

只要看到类的继承，就要考虑子类是否调用的父类的构造器。系统默认调用父类的无参构造器，若父类没有无参构造器，则子类必须自己用super()调用父类的构造器，**并且子类所有的构造器都要调用父类的构造器**。

17.迭代器在使用的过程中，不能对要迭代的集合中的元素进行增加操作，这样虽然编译的时候不会报错，但是在运行的时候会报错。



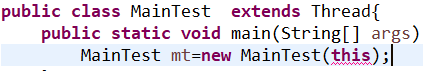
18.Class类没有public修饰的构造函数，其构造函数是private修饰的。Class也是被final修饰的，说明其他任何类都不能继承Class类。





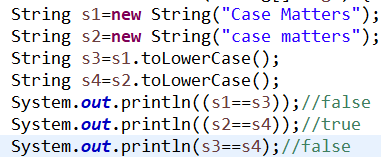
19.case的参数必须是一个常量或一个能够在编译时被评估为常量的表达式。

20.注意在静态方法中不能使用this关键字。因为this关键字指的是当前方法所属的对象，而静态方法是属于类的。

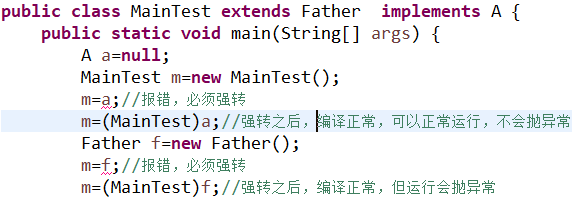
报错

21.**任何修改字符串的方法都不会修改原字符串**

下面代码的结果没弄明白是怎么回事？



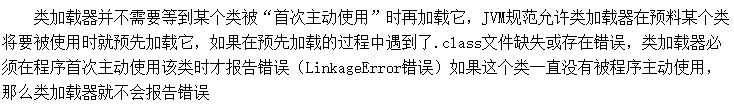
22.注意当子类继承父类，子类实现接口时，若父类new了一个对象，父类引用指向父类对象，然后将父类引用赋值给子类对象必须强转，强转之后编译不会报错，但是运行会报类型转换的异常。若将接口的引用赋值给子类对象，则也必须强转，强转之后运行是不会报异常的，是可以正常运行的。

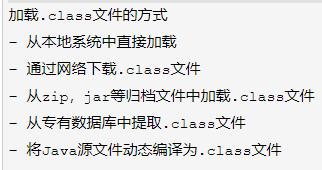


23.java类的加载机制补充：

**类的加载**：指的是将类的.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区中创建一个java.lang.Class对象，用来封装类在方法区内的数据结构。

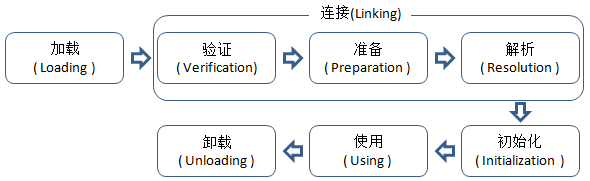
类的加载的最终产物是位于堆区中的Class对象，Class对象封装了类在方法区内的数据结构，并且向java程序员提供了访问方法区内的数据结构的接口。





类加载的过程包括：**加载、验证、准备、解析、初始化**这五个过程。

在这五个阶段中，加载、验证、准备和初始化这四个阶段发生的顺序是确定的，而解析阶段则不一定，它在某些情况下可以在初始化阶段之后开始，这是为了支持Java语言的运行时绑定（也称为动态绑定）。另外注意这里的几个阶段是按顺序开始，而不是按顺序进行或完成，因为这些阶段通常都是互相交叉地混合进行的，通常在一个阶段执行的过程中调用或激活另一个阶段。



**1.加载：查找并加载类的二进制数据**

类加载的第一个阶段就是加载，在加载阶段虚拟机需要完成下面三件事：

1.通过一个类的全限定名来获取其定义的二进制字节流。

2.将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构。

3.在Java堆中生成一个代表这个类的java.lang.Class对象，作为对方法区中这些数据的访问入口。

以使用系统提供的类加载器来完成加载，也可以自定义自己的类加载器来完成加载

加载阶段完成后，虚拟机外部的二进制字节流就按照虚拟机所需的格式存储在方法区之中，而且在Java堆中也创建一个java.lang.Class类的对象，这样便可以通过该对象访问方法区中的这些数据。

**2.连接**：

(1)**验证：确保被加载的类的正确性**

(2)**准备：为类的静态变量分配内存，并将其初始化为默认值**

(3)**解析：把类中的符号引用转换为直接引用**

(1)**验证过程**目的是为了确保Class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。验证阶段非常重要，但不是必须的，它对程序运行其没有影响。

验证阶段大致会完成4个阶段的检验动作：

**文件格式验证**：验证字节流是否符合Class文件格式的规范；

**元数据验证**：对字节码描述的信息进行语义分析（注意：对比javac编译阶段的语义分析），以保证其描述的信息符合Java语言规范的要求

**字节码验证**：通过数据流和控制流分析，确定程序语义是合法的、符合逻辑的。

**符号引用验证**：确保解析动作能正确执行。

(2)**准备阶段**是正式为类变量分配内存并设置类变量初始值的阶段，这些内存都将在方法区中分配。

对于该阶段有以下几点需要注意：

①这时候进行内存分配的仅包括类变量（static），而不包括实例变量，实例变量会在对象实例化时随着对象一块分配在Java堆中。

    ②这里所设置的初始值通常情况下是数据类型默认的值，而不是在Java代码中被显式地赋予的值。

   假设一个类变量的定义为：public static int value = 3，那么变量value在准备阶段过后的初始值为0，而不是3，因为这时候尚未开始执行任何Java方法，而把value赋值为3的putstatic指令是在程序编译后，存放于类构造器<clinit>()方法之中的，所以把value赋值为3的动作将在初始化阶段才会执行。

③**如果类字段的字段属性表中存在ConstantValue属性，即同时被final和static修饰，那么在准备阶段变量value就会被初始化为ConstValue属性所指定的值**。

   假设上面的类变量value被定义为： public static final int value = 3，编译时Javac将会为value生成ConstantValue属性，在准备阶段虚拟机就会根据ConstantValue的设置将value赋值为3。可以理解为static final常量在编译期就将其结果放入了调用它的类的常量池中。

对于同时被static和final修饰的常量，必须在声明的时候就为其显式地赋值，否则编译时不通过；而只被final修饰的常量则既可以在声明时显式地为其赋值，也可以在类初始化时显式地为其赋值，总之，在使用前必须为其显式地赋值，系统不会为其赋予默认零值。

(3)**解析阶段**是虚拟机将常量池内的符号引用替换为直接引用的过程，解析动作主要针对类或接口、字段、类方法、接口方法、方法类型、方法句柄和调用点限定符7类符号引用进行。

**符号引用**就是一组符号来描述目标，可以是任何字面量。

**直接引用**就是直接指向目标的指针、相对偏移量或一个间接定位到目标的句柄。

**3.初始化：为类的静态变量赋予正确的初始值**

在Java中对类变量进行初始值设定有两种方式：

   ①声明类变量时指定初始值

    ②使用静态代码块为类变量指定初始值

JVM初始化步骤：

  ①若这个类还没有被加载和连接，则程序先加载并连接该类

  ②若该类的直接父类还未被初始化，则先初始化其直接父类

  ③若类中有初始化语句，则系统依次执行这些初始化语句

类初始化时机：只有当对类的主动使用的时候才会导致类的初始化，**类的主动使用包括以下六种，即如下六种会使类初始化**：

①创建类的实例，即new的方式

②访问某个类或接口的静态变量，或者对该静态变量赋值

③调用类的静态方法

④反射中的两个获得对象的方法，Class.forName()和getClass

⑤初始化某个类的子类，则其父类也会被初始化

⑥执行main()方法

**结束生命周期**

在如下几种情况下，Java虚拟机将结束生命周期：

①执行了System.exit()方法

②程序正常执行结束

③程序在执行过程中遇到了异常或错误而异常终止

④由于操作系统出现错误而导致Java虚拟机进程终止

**类加载器**

可以大致划分为以下三类

①**启动类加载器**：Bootstrap ClassLoader

②**扩展类加载器**：Extension ClassLoader

③**应用程序类加载器**：Application ClassLoader

**类的加载**

类加载有三种方式：

①命令行启动应用时候由JVM初始化加载

②通过Class.forName()方法动态加载

③通过ClassLoader.loadClass()方法动态加载

**总结：**

**类的加载五个阶段：**

**1.加载：查找并加载类的二进制数据**

**2.连接**：

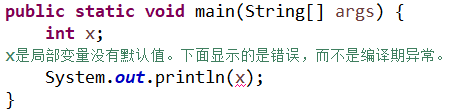
**(1)验证：确保被加载的类的正确性**

**(2)准备：为类的静态变量分配内存,并将其初始化为默认值**

**(3)解析：把类中的符号引用转换为直接引用**

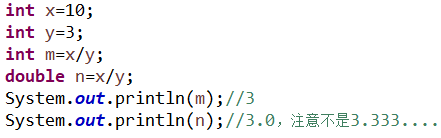
**3.初始化：为类的静态变量赋予正确的初始值**

24. 注意局部变量没有默认值，要使用的话必须赋初始值，否则报错，局部变量就是定义在方法中的变量。全局变量有默认值，是对应的数据类型的默认值，全局变量就是定义在类中的变量。

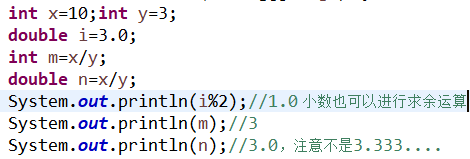


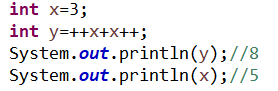
25.只有无符号的右移，没有无符号的左移。无符号右移中，空出来的符号位补0。

26.java中不能将数字进行逻辑运算，即不能用12&&23，但是javaScript中可以。逻辑运算得到的结果为boolean类型。注意java中进行的23&25运算，进行的是按位运算，而不是逻辑运算。



注意小数也可以进行求余运算。





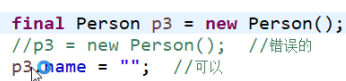
++、--记住只要在后面就后计算，在前面就先计算。上面的表达是4+4=8，然后x再自增为5。

Switch语句中可放String类型的数据，这是从jdk1.7版本之后才能使用的。

**类**：是一个class，是对一类事物的抽象描述。

**对象**：是类的实例，具有属性

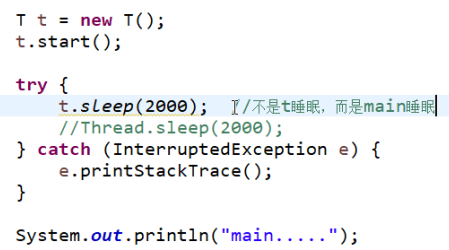
**引用**：是一个变量，通过引用操作对象，一个对象可以有多个引用变量。如果两个引用指向的地址相等，则表示操作的是同一个对象。



上面常量是引用，即p3是常量，不能再指向新的对象。而对象则不是常量，对象中的属性值是可以继续修改的。

static：静态的，修饰的数据是属于类的，因此局部变量不能用static修饰。

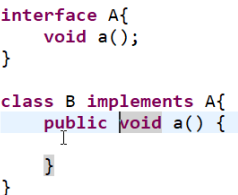
27. 线程中sleep()方法是静态方法，调用该方法要用类。



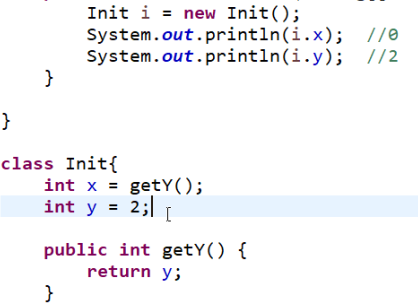
28.在非静态的方法中使用this表当前对象，使用super则是调用父类的内容。**在静态方法中没有this**。

29.静态方法是没有多态的形式，因为静态方法bu能被重写。

30.下面代码中，若B类中的方法不加public则会报错。



31.在对象的初始化中，先给属性赋默认值，即开始时x=0，y=0，之后执行构造块和构造器，开始按顺序初始化属性，在初始化时x=getY()则调用getY()方法进行初始化，原本y为0，因此getY()返回的是0，因此x=0，之后初始化y=2，因此最后x=0，y=2。



32. 线程中若一个类实现Runnable接口，则不是创建了一个新线程，继承Thread则创建的是一个新线程。

33.只有属性，即全局变量才会有线程安全问题，局部变量没有线程安全问题。

34.Hashtable是线程安全的，因为在单线程中不会发生线程安全问题，所以在单线程中尽量不要使用线程安全的Hashtable等，因为会影响效率。

35. 线程中的方法start()、sleep()、join()、yield()。

Object的方法waite()、notify()、notifyAll()。

36.给for循环起别名。

for(){ 循环体1

loop:for(){循环体2

for(){ 循环体3

if(){break loop;}

}

}

}

loop:for(){}是给这个for循环起别名，break loop跳出到指定的循环外，这里跳出到循环体1中继续执行循环。

for(){ 循环体1

for(){ 循环体2

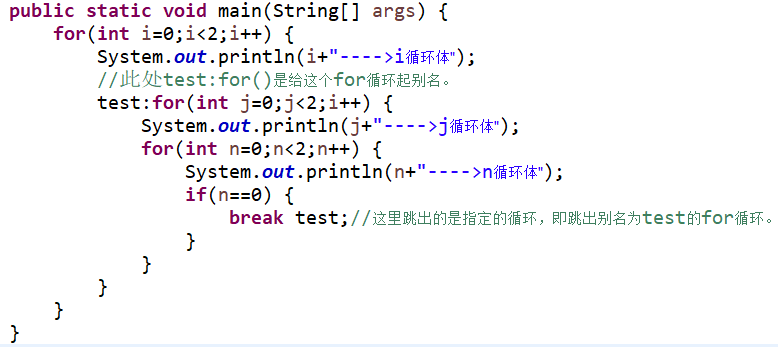
for(){循环体3 break;

}

}

}

这里break跳出的循环体3，然后继续执行循环体2。



37.如何理解java中的面向对象。

面向对象是一种思想，可以理解为将现实世界中的抽象事务看成一个类，这个类中有属性和方法。比如抽象的学生可以定义为一个Student类，而具体的学生比如小明他有学生号、姓名、年龄等信息，他会打球、学习等，而具体的学生小明则可以通过Sutdent类创建一个具体的student对象，这个student对象的属性则封装了学生号、姓名等信息，student对象的方法则封装的是学生小明会做的事情。

38.面向对象的特征有哪些

**封装**：就是将对象的属性进行隐藏，使用private关键字修饰属性，然后对外提供公共方法对属性进行访问，这样可以保证安全性。

**继承**：好处是提高代码的复用性。

**多态**：好处是提高了代码的扩展性。

39.RTTI

运行期类型判定：RTTI(Runtime Type Identification)。

java中存在两种形式的RTTI：传统的RTTI、反射机制。

## 二、this的用法

### **1.this用在构造器中：**

**类中的一个构造器调用另一个构造器用this()，括号内填对应构造器的参数，this必须放在构造器中代码的第一位。**

### **2.this用在成员方法中：**

**指的是当前方法所属的那个对象。**

### 3.不能使用this的地方：

**this关键字不能用在静态区域中，如静态方法中等等。**

## 三、super的用法

### 1.super用在构造器中：

**当子类调用父类的构造器时用super()，括号内填对应的参数。super必须放在构造器中代码的第一位，super和this不能同时使用。**

### 2.super用在成员方法和属性中：

**当子类调用父类的方法或属性时，要借助super，但不能调用父类中被private修饰的属性和方法**。如父类的成员变量a和成员方法smoke()，当在子类中调用时为super.a和super.smoke()。

## 三、final的用法

**final修饰变量表示为常量不能被再次赋值，修饰类则不能被继承，修饰方法则不能被重写**。