JavaOOP面试题

1、什么是B/S架构？什么是C/S架构

1. B/S(Browser/Server)，浏览器/服务器程序
2. C/S(Client/Server)，客户端/服务端，桌面应用程序

2、Java都有那些开发平台？

1. JAVA SE：主要用在客户端开发
2. JAVA EE：主要用在web应用程序开发
3. JAVA ME：主要用在嵌入式应用程序开发

3、什么是JDK？什么是JRE?

1. JDK：java development kit：java开发工具包，是开发人员所需要安装的环境
2. JRE：java runtime environment：java运行环境，java程序运行所需要安装的环境

4、Java语言有哪些特点

1. 简单易学、有丰富的类库
2. 面向对象（Java最重要的特性，让程序耦合度更低，内聚性更高）
3. 与平台无关性（JVM是Java跨平台使用的根本）
4. 可靠安全
5. 支持多线程

5、面向对象和面向过程的区别

1. 面向过程：

一种较早的编程思想，顾名思义就是该思想是站着过程的角度思考问题，强调的就是功能行为，功能的执行过程，即先后顺序，而每一个功能我们都使用函数（类似于方法）把这些步骤一步一步实现。使用的时候依次调用函数就可以了。

2. 面向对象： 一种基于面向过程的新编程思想，顾名思义就是该思想是站在对象的角度思考问题，我们把多个功能合理放到不同对象里，强调的是具备某些功能的对象。

   具备某种功能的实体，称为对象。面向对象最小的程序单元是：类。面向对象更加符合常规的思维方式，稳定性好，可重用性强，易于开发大型软件产品，有良好的可维护性。

   在软件工程上，面向对象可以使工程更加模块化，实现更低的耦合和更高的内聚。

6、什么是数据结构？

计算机保存，组织数据的方式

7、Java的数据结构有那些？

1.线性表（ArrayList） 2.链表（LinkedList） 3.栈（Stack） 4.队列（Queue） 5.图（Map） 6.树（Tree）

8、什么是OOP?

面向对象编程

9、类与对象的关系?

类是对象的抽象，对象是类的具体，类是对象的模板，对象是类的实例

10、Java中有几种数据类型

整形：byte,short,int,long 浮点型：float,double 字符型：char 布尔型：boolean

11、标识符的命名规则。

1. 标识符的含义：

   是指在程序中，我们自己定义的内容，譬如，类的名字，方法名称以及变量名称等等，都是标识符。

2. 命名规则：（硬性要求）

   标识符可以包含英文字母，0-9的数字，$以及_

   标识符不能以数字开头

   标识符不是关键字

3. 命名规范：（非硬性要求）

   类名规范：首字符大写，后面每个单词首字母大写（大驼峰式）。

变量名规范：首字母小写，后面每个单词首字母大写（小驼峰式）。

方法名规范：同变量名。

12、instanceof关键字的作用

instanceof 严格来说是Java中的一个双目运算符，用来测试一个对象是否为一个类的实例，用法为：

```
boolean result = obj instanceof Class
```

其中 obj 为一个对象，Class 表示一个类或者一个接口，当 obj 为 Class 的对象，或者是其直接或间接子类，或者是其接口的实现类，结果result 都返回 true，否则返回false。

注意：编译器会检查 obj 是否能转换成右边的class类型，如果不能转换则直接报错，如果不能确定类型，则通过编译，具体看运行时定。

```
inti=0;

System.out.println(i instanceof Integer);//编译不通过i必须是引用类型，不能是基本类型

System.out.println(i instanceof Object);//编译不通过

Integer integer=newInteger(1);

System.out.println(integer instanceof Integer);//true
//false,在JavaSE规范中对instanceof运算符的规定就是：如果obj为null，那么将返回false。
System.out.println(nullinstanceofObject);
```

13、什么是隐式转换，什么是显式转换

显示转换就是类型强转，把一个大类型的数据强制赋值给小类型的数据；隐式转换就是大范围的变量能够接受小范围的数据；隐式转换和显式转换其实就是自动类型转换和强制类型转换。

14、Char类型能不能转成int类型？能不能转化成string类型，能不能转成double类型

Char在java中也是比较特殊的类型，它的int值从1开始，一共有2的16次方个数据；

Char<int<long<float<double；Char类型可以隐式转成int,double类型，但是不能隐式转换成string；如果char类

型转成byte，short类型的时候，需要强转。

15、什么是拆装箱？

1. 装箱就是自动将基本数据类型转换为包装器类型（int-->Integer）；调用方法：Integer的**valueOf(int) 方法**

拆箱就是自动将包装器类型转换为基本数据类型（Integer-->int）。调用方法：Integer的intValue方 法

在Java SE5之前，如果要生成一个数值为10的Integer对象，必须这样进行：

```
Integer i = new Integer(10);
```

而在从Java SE5开始就提供了自动装箱的特性，如果要生成一个数值为10的Integer对象，只需要这

样就可以了：

```
Integer i = 10;
```

面试题**1**： 以下代码会输出什么？

```
public class Main {

    public static void main(String[] args) {

        Integer i1 = 100;

        Integer i2 = 100;

        Integer i3 = 200;

        Integer i4 = 200;

        System.out.println(i1==i2);

        System.out.println(i3==i4);
```

```

    }

}
```

结果：

```
true

false
```

16、Java中的包装类都是那些？

byte：Byte，short：Short，int：Integer，long：Long，float：Float，double：Double，char：Character ，boolean：Boolean

17、一个java类中包含那些内容？

属性、方法、内部类、构造方法、代码块。

18、那针对浮点型数据运算出现的误差的问题，你怎么解决？

使用Bigdecimal类进行浮点型数据的运算

19、面向对象的特征有哪些方面?

抽象:

抽象是将一类对象的共同特征总结出来构造类的过程, 包括数据抽象和行为抽象两方面。抽象只关注对象有哪些属

性和行为,并不关注这些行为的细节是什么。

继承:

继承是从已有类得到继承信息创建新类的过程.提供继承信息的类被称为父类(超类、基类) ;得到继承信息的类被称

为子类(派生类)。继承让变化中的软件系统有了一定的延续性 ,同时继承也是封装程序中可变因素的重要手段(如果

不能理解请阅读阎宏博士的《Java 与模式》或《设计模式精解》中.关于桥梁模式的部分)。

封装：

通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

多态性：

多态性是指允许不同子类型的对象对同一消息作出不同的响应。

简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当 A 系统访问 B 系统提供的服务时，B 系统有多种提供服务的方式，但一切对 A 系统来说都是透明的（就像电动剃须刀是 A 系统，它的供电系统是 B 系统，B 系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A 系统只会通过 B 类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1). 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2). 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

20、访问修饰符 public,private,protected,以及不写（默认）时的区别？

修饰符	当前类	同 包	子 类	其他包
public	能	能	能	能
protected	能	能	能	不能
default	能	能	不能	不能
private	能	不能	不能	不能

类的成员不写访问修饰时默认为 default。默认对于同一个包中的其他类相当于公 开（public），对于不是同一个包中的其他类相当于私有（private）。受保护 （protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私 有。Java 中，外部类的修饰符只能是 public 或默认，类的成员（包括内部类）的 修饰符可以是以上四种。

21、String 是最基本的数据类型吗？

不是。Java 中的基本数据类型只有 8 个：byte、short、int、long、float、double、 char、boolean；除了基本类型（primitive type），剩下的都是引用类型（reference type），Java 5 以后引入的枚举类型也算是一种比较特殊的引用类型。

22、float f=3.4;是否正确？

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于 下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换 float f =(float)3.4; 或者写成 float f =3.4F;。

23、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1; 有错吗？

对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1 += 1;可以正确 编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

24、重载和重写的区别

重写(Override)**

从字面上看，重写就是 重新写一遍的意思。其实就是在子类中把父类本身有的方法重新写一遍。子类继承了父类

原有的方法，但有时子类并不想原封不动的继承父类中的某个方法，所以在方法名，参数列表，返回类型(除过子

类中方法的返回值是父类中方法返回值的子类时)都相同的情况下， 对方法体进行修改或重写，这就是重写。但要

注意子类函数的访问修饰权限不能少于父类的。

```
public class Father {

    public static void main(String[] args) {

        // TODO Auto-generated method stub

        Son s = new Son();

        s.sayHello();

    }

    public void sayHello() {

        System.out.println("Hello");
    }
```

```
}

class Son extends Father{

    @Override

    public void sayHello() {

        // TODO Auto-generated method stub

        System.out.println("hello by ");

    }

}
```

原因： 在某个范围内的整型数值的个数是有限的，而浮点数却不是。

重写 总结：

1.发生在父类与子类之间

2.方法名，参数列表，返回类型（除过子类中方法的返回类型是父类中返回类型的子类）必须相同

3.访问修饰符的限制一定要大于被重写方法的访问修饰符（public>protected>default>private)

4.重写方法一定不能抛出新的检查异常或者比被重写方法申明更加宽泛的检查型异常

重载（Overload）

在一个类中，同名的方法如果有不同的参数列表（**参数类型不同、参数个数不同甚至是参数顺序不同**）则视为重载。同时，重载对返回类型没有要求，可以相同也可以不同，但**不能通过返回类型是否相同来判断重载。**

```
public class Father {

    public static void main(String[] args) {

        // TODO Auto-generated method stub

        Father s = new Father();

        s.sayHello();

        s.sayHello("wintershii");

      }

    public void sayHello() {

        System.out.println("Hello");

    }
```

```
    public void sayHello(String name) {
        System.out.println("Hello" + " " + name);
    }
}
```

重载总结：

1.重载Overload是一个类中多态性的一种表现

2.重载要求同名方法的参数列表不同(参数类型，参数个数甚至是参数顺序)

3.重载的时候，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准

25、equals与==的区别

==：

== 比较的是变量(栈)内存中存放的对象的(堆)内存地址，用来判断两个对象的地址是否相同，即是否是指相同一个对象。比较的是真正意义上的指针操作。

1、比较的是操作符两端的操作数是否是同一个对象。

2、两边的操作数必须是同一类型的（可以是父子类之间）才能编译通过。

3、比较的是地址，如果是具体的阿拉伯数字的比较，值相等则为true，如：

int a=10 与 long b=10L 与 double c=10.0都是相同的（为true），因为他们都指向地址为10的堆。

equals：

equals用来比较的是两个对象的内容是否相等，由于所有的类都是继承自java.lang.Object类的，所以适用于所有对象，如果没有对该方法进行覆盖的话，调用的仍然是Object类中的方法，而Object中的equals方法返回的却是==的判断。

总结：

所有比较是否相等时，都是用equals 并且在对常量相比较时，把常量写在前面，因为使用object的equals object可能为null 则空指针

在阿里的代码规范中只使用equals ，阿里插件默认会识别，并可以快速修改，推荐安装阿里插件来排查老代码使用“==”，替换成equals

36、++i与i++的区别

i++：先赋值，后计算 ++i：先计算，后赋值

37、程序的结构有那些？

```
顺序结构
选择结构
循环结构
```

38、数组实例化有几种方式？

静态实例化：创建数组的时候已经指定数组中的元素,

```
int [] a= new int[]{ 1 , 3 , 3}
```

动态实例化：实例化数组的时候，只指定了数组程度，数组中所有元素都是数组类型的默认值

39、Java中各种数据默认值

Byte,short,int,long默认是都是0

Boolean默认值是false

Char类型的默认值是”

Float与double类型的默认是0.0

对象类型的默认值是null

40、Java常用包有那些？

```
Java.lang
Java.io
Java.sql
Java.util
Java.awt
Java.net
Java.math
```

41、Object类常用方法有那些？

```
Equals
Hashcode
toString
wait
notify
clone
getClass
```

42、java中有没有指针？

有指针，但是隐藏了，开发人员无法直接操作指针，由jvm来操作指针

43、java中是值传递引用传递？

理论上说，java都是引用传递，对于基本数据类型，传递是值的副本，而不是值本身。对于对象类型，传递是对象的引用，当在一个方法操作操作参数的时候，其实操作的是引用所指向的对象。

44、实例化数组后，能不能改变数组长度呢？

不能，数组一旦实例化，它的长度就是固定的

45、假设数组内有5个元素，如果对数组进行反序，该如何做？

创建一个新数组，从后到前循环遍历每个元素，将取出的元素依次顺序放入新数组中

46、形参与实参区别

实参(argument)：

全称为"实际参数"是在调用时传递给函数的参数. 实参可以是常量、变量、表达式、函数等， 无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值， 以便把这些值传送给形参。 因此应预先用赋值，输入等办法使实参获得确定值。

形参(parameter)：

全称为"形式参数" 由于它不是实际存在变量，所以又称虚拟变量。是在定义函数名和函数体的时候使用的参数,目的是用来接收调用该函数时传入的参数.在调用函数时，实参将赋值给形参。因而，必须注意实参的个数，类型应与形参一一对应，并且实参必须要有确定的值。

形参出现在**函数定义**中，在整个函数体内都可以使用， 离开该函数则不能使用。

实参出现在**主调函数中，进入被调函数后，实参变量也不能使用。**

形参和实参的功能是作数据传送。发生函数调用时， **主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。**

1.形参变量只有在被调用时才分配内存单元，**在调用结束时， 即刻释放所分配的内存单元。**因此，形参只有在函数内部有效。 函数调用结束返回

主调函数后则不能再使用该形参变量。

2.实参可以是常量、变量、表达式、函数等， 无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值， 以便把这些值传送给形

参。 因此应预先用赋值，输入等办法使实参获得确定值。

3.实参和形参在数量上，类型上，顺序上应严格一致， 否则会发生“类型不匹配”的错误。

4.**函数调用中发生的数据传送是单向的。** 即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。 因此在函数调用过程中，形参的值

发生改变，而实参中的值不会变化。

5.当形参和实参不是指针类型时，在该函数运行时，**形参和实参是不同的变量，他们在内存中位于不同的位置，形参将实参的内容复制一份，在该**

函数运行结束的时候形参被释放，而实参内容不会改变。

而**如果函数的参数是指针类型变量,在调用该函数的过程中，传给函数的是实参的地址，在函数体内部使用的也是实参的地址，即使用的就是实参**

本身。所以在函数体内部可以改变实参的值。

47、构造方法能不能显式调用？

不能，构造方法当成普通方法调用，只有在创建对象的时候它才会被系统调用

48、什么是方法重载？

方法的重载就是在同一个类中允许同时存在一个以上的同名方法，只要它们的参数个数或者类型不同即可。在这种情况下，该方法就叫被重载了，这个过程称为方法的重载（override）

49、构造方法能不能重写？能不能重载？

可以重载，但不能重写。

50、内部类与静态内部类的区别？

静态内部类相对与外部类是独立存在的，在静态内部类中无法直接访问外部类中变量、方法。如果要访问的话，必须要new一个外部类的对象，使用new出来的对象来访问。但是可以直接访问静态的变量、调用静态的方法；

普通内部类作为外部类一个成员而存在，在普通内部类中可以直接访问外部类属性，调用外部类的方法。

如果外部类要访问内部类的属性或者调用内部类的方法，必须要创建一个内部类的对象，使用该对象访问属性或者调用方法。

如果其他的类要访问普通内部类的属性或者调用普通内部类的方法，必须要在外部类中创建一个普通内部类的对象作为一个属性，外同类可以通过该属性调用普通内部类的方法或者访问普通内部类的属性

如果其他的类要访问静态内部类的属性或者调用静态内部类的方法，直接创建一个静态内部类对象即可。

51、Static关键字有什么作用？

Static可以修饰内部类、方法、变量、代码块

Static修饰的类是静态内部类

Static修饰的方法是静态方法，表示该方法属于当前类的，而不属于某个对象的，静态方法也不能被重写，可以直接使用类名来调用。在static方法中不能使用this或者super关键字。

Static修饰变量是静态变量或者叫类变量，静态变量被所有实例所共享，不会依赖于对象。静态变量在内存中只有一份拷贝，在JVM加载类的时候，只为静态分配一次内存。

Static修饰的代码块叫静态代码块，通常用来做程序优化的。静态代码块中的代码在整个类加载的时候只会执行一次。静态代码块可以有多个，如果有多个，按照先后顺序依次执行。

52、final在java中的作用，有哪些用法?

final也是很多面试喜欢问的地方,但我觉得这个问题很无聊,通常能回答下以下5点就不错了:

1. 被fifinal修饰的类不可以被继承
2. 被fifinal修饰的方法不可以被重写
3. 被fifinal修饰的变量不可以被改变.如果修饰引用,那么表示引用不可变,引用指向的内容可变.
4. 被fifinal修饰的方法,JVM会尝试将其内联,以提高运行效率
5. 被fifinal修饰的常量,在编译阶段会存入常量池中.

除此之外,编译器对fifinal域要遵守的两个重排序规则更好:

在构造函数内对一个fifinal域的写入,与随后把这个被构造对象的引用赋值给一个引用变量,这两个操作之间不能重排序

初次读一个包含fifinal域的对象的引用,与随后初次读这个fifinal域,这两个操作之间不能重排序

53、StringString StringBuffffer 和 StringBuilder 的区别是什么？

String是只读字符串，它并不是基本数据类型，而是一个对象。从底层源码来看是一个fifinal类型的字符

数组，所引用的字符串不能被改变，一经定义，无法再增删改。每次对String的操作都会生成新的

String对象

```
private final char value[];
```

每次+操作 ： 隐式在堆上new了一个跟原字符串相同的StringBuilder对象，再调用append方法 拼接

+后面的字符。

StringBuffer与StringBuilder都继承了AbstractStringBulder类，而AbtractStringBuilder又实现了CharSequence接口，两个类都是用来进行字符串操作的。

在做字符串拼接修改删除替换时，效率比string更高。

StringBuffer是线程安全的，Stringbuilder是非线程安全的。所以Stringbuilder比stringbuffer效率更高，StringBuffer的方法大多都加了synchronized关键字

54、String str=”aaa”,与String str=new String(“aaa”)一样吗？

一共有两个引用，三个对象。因为”aa”与”bb”都是常量，常量的值不能改变，当执行字符串拼接时候，会创建一个新的常量是” aabbb”,有将其存到常量池中。

55、讲下java中的math类有那些常用方法？

Pow()：幂运算 Sqrt()：平方根 Round()：四舍五入 Abs()：求绝对值 Random()：生成一个0-1的随机数，包括0不包括1

56、String类的常用方法有那些？

charAt：返回指定索引处的字符 indexOf()：返回指定字符的索引 replace()：字符串替换 trim()：去除字符串两端空白 split()：分割字符串，返回一个分割后的字符串数组 getBytes()：返回字符串的byte类型数组 length()：返回字符串长度 toLowerCase()：将字符串转成小写字母 toUpperCase()：将字符串转成大写字符 substring()：截取字符串 format()：格式化字符串 equals()：字符串比较

57、Java中的继承是单继承还是多继承

Java中既有单继承，又有多继承。对于java类来说只能有一个父类，对于接口来说可以同时继承多个接口

58、Super与this表示什么？

Super表示当前类的父类对象 This表示当前类的对象

59、普通类与抽象类有什么区别？

普通类不能包含抽象方法，抽象类可以包含抽象方法 抽象类不能直接实例化，普通类可以直接实例化

60、什么是接口？为什么需要接口？

接口就是某个事物对外提供的一些功能的声明，是一种特殊的java类，接口弥补了java单继承的缺点

61、接口有什么特点？

接口中声明全是public static final修饰的常量 接口中所有方法都是抽象方法 接口是没有构造方法的 接口也不能直接实例化 接口可以多继承

62、抽象类和接口的区别?

抽象类：

1. 抽象方法，只有行为的概念，没有具体的行为实现。使用abstract关键字修饰，没有方法体。子类必须重写这些抽象方法。

2. 包含抽象方法的类，一定是抽象类。
3. 抽象类只能被继承，一个类只能继承一个抽象类。

接口：

1. 全部的方法都是抽象方法，属性都是常量
2. 不能实例化，可以定义变量。
3. 接口变量可以引用具体实现类的实例
4. 接口只能被实现，一个具体类实现接口，必须实现全部的抽象方法
5. 接口之间可以多实现
6. 一个具体类可以实现多个接口，实现多继承现象

63、Hashcode的作用

java的集合有两类，一类是List，还有一类是Set。前者有序可重复，后者无序不重复。当我们在set中插入的时候怎么判断是否已经存在该元素呢，可以通过equals方法。但是如果元素太多，用这样的方法就会比较满。

于是有人发明了哈希算法来提高集合中查找元素的效率。 这种方式将集合分成若干个存储区域，每个对象可以计算出一个哈希码，可以将哈希码分组，每组分别对应某个存储区域，根据一个对象的哈希码就可以确定该对象应该存储的那个区域。

hashCode方法可以这样理解：它返回的就是根据对象的内存地址换算出的一个值。这样一来，当集合要添加新的元素时，先调用这个元素的hashCode方法，就一下子能定位到它应该放置的物理位置上。如果这个位置上没有元素，它就可以直接存储在这个位置上，不用再进行任何比较了；如果这个位置上已经有元素了，就调用它的equals方法与新元素进行比较，相同的话就不存了，不相同就散列其它的地址。这样一来实际调用equals方法的次数就大大降低了，几乎只需要一两次。

64、 Java的四种引用，强弱软虚

强引用

强引用是平常中使用最多的引用，强引用在程序内存不足（OOM）的时候也不会被回收，使用方式：

```
String str = new String("str");
```

软引用

软引用在程序内存不足时，会被回收，使用方式：

```
// 注意：wrf这个引用也是强引用，它是指向SoftReference这个对象的，

// 这里的软引用指的是指向new String("str")的引用，也就是SoftReference类中T

SoftReference<String> wrf = new SoftReference<String>(new String("str"));
```

可用场景： 创建缓存的时候，创建的对象放进缓存中，当内存不足时，JVM就会回收早先创建的对象。

弱引用

弱引用就是只要JVM垃圾回收器发现了它，就会将之回收，使用方式：

```
WeakReference<String>wrf=newWeakReference<String>(str);
```

可用场景：Java源码中的java.util.WeakHashMap中的key就是使用弱引用，我的理解就是，

一旦我不需要某个引用，JVM会自动帮我处理它，这样我就不需要做其它操作。

虚引用

虚引用的回收机制跟弱引用差不多，但是它被回收之前，会被放入ReferenceQueue中。注意哦，其它引用是被JVM回收后才被传入ReferenceQueue中的。由于这个机制，所以虚引用大多被用于引用销毁前的处理工作。还有就是，虚引用创建的时候，必须带有ReferenceQueue，使用

例子：

```
PhantomReference<String>prf=newPhantomReference<String>(new

String("str"),newReferenceQueue<>());
```

可用场景： 对象销毁前的一些操作，比如说资源释放等。** Object.finalize() 虽然也可以做这类动作，但是这个方式即不安全又低效

上诉所说的几类引用，都是指对象本身的引用，而不是指 Reference **的四个子类的引用**

(SoftReference 等)。

65、Java创建对象有几种方式？

java中提供了以下四种创建对象的方式:

1. new创建新对象
2. 通过反射机制
3. 采用clone机制
4. 通过序列化机制

66、有没有可能两个不相等的对象有相同的hashcode

有可能.在产生hash冲突时,两个不相等的对象就会有相同的 hashcode 值.当hash冲突产生时,一般有以

下几种方式来处理:

1. 拉链法:每个哈希表节点都有一个next指针,多个哈希表节点可以用next指针构成一个单向链表，被分配到同一个索引上的多个节点可以用这个单向链表进行存储.
2. 开放定址法:一旦发生了冲突,就去寻找下一个空的散列地址,只要散列表足够大,空的散列地址总能找到,并将记录存入
3. 再哈希:又叫双哈希法,有多个不同的Hash函数.当发生冲突时,使用第二个,第三个....等哈希函数计算地址,直到无冲突.

67、拷贝和浅拷贝的区别是什么?

浅拷贝:

被复制对象的所有变量都含有与原来的对象相同的值,而所有的对其他对象的引用仍然指向原来的对象.换言之,浅拷贝仅仅复制所考虑的对象,而不复制它所引用的对象.

深拷贝:

被复制对象的所有变量都含有与原来的对象相同的值.而那些引用其他对象的变量将指向被复制过的新对象.而不再是原有的那些被引用的对象.换言之.深拷贝把要复制的对象所引用的对象都复制了一遍.

68、static都有哪些用法?

所有的人都知道static关键字这两个基本的用法:静态变量和静态方法.也就是被static所修饰的变量/方法都属于类的静态资源,类实例所共享.

除了静态变量和静态方法之外,static也用于静态块,多用于初始化操作：

```
public calss PreCache{

    static{

    //执行相关操作

    }

}
```

此外static也多用于修饰内部类,此时称之为静态内部类.

最后一种用法就是静态导包,即 import static .import static是在JDK 1.5之后引入的新特性,可以用来指定导入某个类中的静态资源,并且不需要使用类名,可以直接使用资源名,比如:

```
import static java.lang.Math.*;

public class Test{

    public static void main(String[] args){

        //System.out.println(Math.sin(20));传统做法

        System.out.println(sin(20));

    }

}
```

69、a=a+b与a+=b有什么区别吗?

+= 操作符会进行隐式自动类型转换,此处a+=b隐式的将加操作的结果类型强制转换为持有结果的类型,而a=a+b则不会自动进行类型转换.如：

```
byte a = 127;

byte b = 127;

b = a + b; // 报编译错误:cannot convert from int to byte

b += a;
```

以下代码是否有错,有的话怎么改？

```
short s1= 1;

s1 = s1 + 1;
```

有错误.short类型在进行运算时会自动提升为int类型,也就是说 s1+1 的运算结果是int类型,而s1是short类型,此时编译器会报错.

正确写法：

```
short s1= 1;

s1 += 1;
```

+=操作符会对右边的表达式结果强转匹配左边的数据类型,所以没错.

70、final、finalize()、finally

性质不同

1. final为关键字；
2. finalize()为方法；
3. finally为区块标志，用于try语句中；

作用

1. final为用于标识常量的关键字，final标识的关键字存储在常量池中（在这里final常量的具体用法将在下面进行介绍）；
2. finalize()方法在Object中进行了定义，用于在对象“消失”时，由JVM进行调用用于对对象进行垃圾回收，类似于C++中的析构函数；用户自定义时，用于释放对象占用的资源（比如进行I/0操作）；
3. finally{}用于标识代码块，与try{}进行配合，不论try中的代码执行完或没有执行完（这里指有异常），该代码块之中的程序必定会进行；

71、JDBC操作的步骤

加载数据库驱动类 打开数据库连接 执行sql语句 处理返回结果 关闭资源

72、在使用jdbc的时候，如何防止出现sql注入的问题。

使用PreparedStatement类，而不是使用Statement类

73、怎么在JDBC内调用一个存储过程

使用CallableStatement

74、是否了解连接池，使用连接池有什么好处？

数据库连接是非常消耗资源的，影响到程序的性能指标。连接池是用来分配、管理、释放数据库连接的，可以使应用程序重复使用同一个数据库连接，而不是每次都创建一个新的数据库连接。通过释放空闲时间较长的数据库连接避免数据库因为创建太多的连接而造成的连接遗漏问题，提高了程序性能。

75、你所了解的数据源技术有那些？使用数据源有什么好处？

Dbcp,c3p0等，用的最多还是c3p0，因为c3p0比dbcp更加稳定，安全；通过配置文件的形式来维护数据库信息，而不是通过硬编码。当连接的数据库信息发生改变时，不需要再更改程序代码就实现了数据库信息的更新。

76、&和&&的区别

&是位运算符。&&是布尔逻辑运算符，在进行逻辑判断时用&处理的前面为false后面的内容仍需处理，用&&处理的前面为false不再处理后面的内容。

77、静态内部类如何定义

定义在类内部的静态类，就是静态内部类。

```
public class Out {

     private static int a;

     private int b;

     public static class Inner {

          public void print() {

          System.out.println(a);

          }

     }

}
```

1. 静态内部类可以访问外部类所有的静态变量和方法，即使是 private 的也一样。

2. 静态内部类和一般类一致，可以定义静态变量、方法，构造方法等。
3. 其它类使用静态内部类需要使用“外部类.静态内部类”方式，如下所示：Out.Inner inner = new Out.Inner();inner.print();
4. Java集合类HashMap内部就有一个静态内部类Entry。Entry是HashMap存放元素的抽象，HashMap 内部维护 Entry 数组用了存放元素，但是 Entry 对使用者是透明的。像这种和外部类关系密切的，且不依赖外部类实例的，都可以使用静态内部类。

78、什么是成员内部类

定义在类内部的非静态类，就是成员内部类。成员内部类不能定义静态方法和变量（final修饰的除外）。这是因为成员内部类是非静态的，类初始化的时候先初始化静态成员，如果允许成员内部类定义静态变量，那么成员内部类的静态变量初始化顺序是有歧义的。实例：

```
public class Out {

    private static int a;

    private int b;

    public class Inner {

        public void print() {

            System.out.println(a);

            System.out.println(b);

        }

    }

}
```

79、Static Nested Class 和 Inner Class的不同

Nested Class （一般是C++的说法），Inner Class (一般是JAVA的说法)。Java内部类与C++嵌套类最大的不同就在于是否有指向外部的引用上。注： 静态内部类（Inner Class）意味着1创建一个static内部类的对象，不需要一个外部类对象，2不能从一个static内部类的一个对象访问一个外部类对象

80、什么时候用assert

assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。在实现中，assertion就是在程序中的一条语句，它对一个boolean表达式进行检查，一个正确程序必须保证这个boolean表达式的值为true；如果该值为false，说明程序已经处于不正确的状态下，系统将给出警告或退出。一般来说，assertion用于保证程序最基本、关键的正确性。assertion检查通常在开发和测试时开启。为了提高性能，在软件发布后，assertion检查通常是关闭的

81、Java有没有goto

java中的保留字，现在没有在java中使用

82、数组有没有length()这个方法? String有没有length()这个方法

数组没有length()这个方法，有length的属性。String有有length()这个方法

83、用最有效率的方法算出2乘以8等於几

2 << 3

84、float型float f=3.4是否正确?

不正确。精度不准确,应该用强制类型转换，如下所示：float f=(float)3.4

85、排序都有哪几种方法？请列举

排序的方法有：插入排序（直接插入排序、希尔排序），交换排序（冒泡排序、快速排序），选择排序（直接选择排序、堆排序），归并排序，分配排序（箱排序、基数排序）快速排序的伪代码。/ /使用快速排序方法对a[0 :n- 1]排序从a[0 :n- 1]中选择一个元素作为m i d d l e，该元素为支点把余下的元素分割为两段left 和r i g h t，使得l e f t中的元素都小于等于支点，而right 中的元素都大于等于支点递归地使用快速排序方法对left 进行排序递归地使用快速排序方法对right 进行排序所得结果为l e f t + m i d d l e + r i g h t

86、静态变量和实例变量的区别？

static i = 10; //常量 class A a; a.i =10;//可变

87、说出一些常用的类，包，接口，请各举5个

常用的类：BufferedReader BufferedWriter FileReader FileWirter String Integer常用的包：java.lang java.awt java.io java.util java.sql常用的接口：Remote List Map Document NodeList

88、a.hashCode() 有什么用？与 a.equals(b) 有什么关系？

hashCode() 方法是相应对象整型的 hash 值。它常用于基于 hash 的集合类，如 Hashtable、HashMap、LinkedHashMap 等等。它与 equals() 方法关系特别紧密。根据 Java 规范，两个使用 equal() 方法来判断相等的对象，必须具有相同的 hash code。

89、Java 中的编译期常量是什么？使用它又什么风险？

公共静态不可变（public static final ）变量也就是我们所说的编译期常量，这里的 public 可选的。实际上这些变量在编译时会被替换掉，因为编译器知道这些变量的值，并且知道这些变量在运行时不能改变。这种方式存在的一个问题是你使用了一个内部的或第三方库中的公有编译时常量，但是这个值后面被其他人改变了，但是你的客户端仍然在使用老的值，甚至你已经部署了一个新的 jar。为了避免这种情况，当你在更新依赖 JAR 文件时，确保重新编译你的程序

90、在 Java 中，如何跳出当前的多重嵌套循环？

在最外层循环前加一个标记如 A，然后用 break A;可以跳出多重循环。（Java 中支持带标签的 break 和 continue 语句，作用有点类似于 C 和 C++中的 goto 语句，但是就像要避免使用 goto 一样，应该避免使用带标签的 break 和 continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

91、构造器（constructor）是否可被重写（override）？

构造器不能被继承，因此不能被重写，但可以被重载。

92、两个对象值相同(x.equals(y) == true)，但却可有不同的hash code，这句话对不对？

不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的：

(1)如果两个 对象相同（equals 方法返回 true），那么它们的 hashCode 值一定要相同；

(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

93、是否可以继承 String 类？

String 类是 final 类，不可以被继承，继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关 联关系（Has-A）和依赖关系（Use-A）而不是继承关系（Is-A）。

94、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？

是值传递。Java 语言的方法调用只支持参数的值传递。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象的属性可以在被调用过程中被改变，但对对象引用的改变是不会影响到调用者的。C++和 C#中可以通过传引用或传输出参数来改变传入的参数的值。在 C#中可以编写如下所示的代码，但是在 Java 中却做不到。

```
using System;
namespace CS01 {
    class Program {
```

```
        public static void swap(ref int x, ref int y) {
            int temp = x;
            x = y;
            y = temp;
        } p
        ublic static void Main (string[] args) {
            int a = 5, b = 10;
            swap (ref a, ref b);
// a = 10, b = 5;
            Console.WriteLine ("a = {0}, b = {1}", a, b);
        }
    }
}
```

说明：Java 中没有传引用实在是非常的不方便，这一点在 Java 8 中仍然没有得到改进，正是如此在 Java 编写的代码中才会出现大量的 Wrapper 类（将需要通过方法调用修改的引用置于一个 Wrapper 类中，再将 Wrapper 对象传入方法），这样的做法只会让代码变得臃肿，尤其是让从 C 和 C++转型为 Java 程序员的开发者无法容忍。

95、String 和 StringBuilder、StringBuffer 的区别？

Java 平台提供了两种类型的字符串：String 和 StringBuffer/StringBuilder，它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。而 StringBuffer/StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 Java 5 中引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被synchronized 修饰，因此它的效率也比 StringBuffer 要高。

96、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

97、char 型变量中能不能存贮一个中文汉字，为什么？

char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个 char 类型占 2 个字节（16 比特），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于 C 程序员来说，要完成这样的编码转换恐怕要依赖于 union（联合体/共用体）共享内存的特征来实现了。

98、抽象类（abstract class）和接口（interface）有什么异同 ？

抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是 private、默认、protected、public 的，而接口中的成员全都是 public 的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

99、静态嵌套类(Static Nested Class)和内部类（Inner Class）的不同？

Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实例化后才能实例化，其语法看起来挺诡异的，如下所示

```
/**
* 扑克类（一副扑克）
* @author 骆昊
*/
public class Poker {
    private static String[] suites = {"黑桃", "红桃", "草花", "方块"};
    private static int[] faces = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
    private Card[] cards;
    /**
    * 构造器
    * */
    public Poker() {
        cards = new Card[52];
        for(int i = 0; i < suites.length; i++) {
            for(int j = 0; j < faces.length; j++) {
                cards[i * 13 + j] = new Card(suites[i], faces[j]);
            }
        }
    }
    /**
    * 洗牌 （随机乱序）
    * */
    public void shuffle() {
        for(int i = 0, len = cards.length; i < len; i++) {
            int index = (int) (Math.random() * len);
            Card temp = cards[index];
            cards[index] = cards[i];
            cards[i] = temp;
        }
    }
    /**
    * 发牌
    * @param index 发牌的位置
    * */
    public Card deal(int index) {
        return cards[index];
    }
```

```
    /**
    * 卡片类（一张扑克）
    * [内部类]
    * @author 骆昊
    * */
    public class Card {
        private String suite; // 花色
        private int face; // 点数
        public Card(String suite, int face) {
            this.suite = suite;
            this.face = face;
        }
        @Override
        public String toString() {
            String faceStr = "";
            switch(face) {
                case 1: faceStr = "A"; break;
                case 11: faceStr = "J"; break;
                case 12: faceStr = "Q"; break;
                case 13: faceStr = "K"; break;
                default: faceStr = String.valueOf(face);
            } return suite + faceStr;
        }
    }
}

//测试类
class PokerTest {
    public static void main(String[] args) {
        Poker poker = new Poker();
        poker.shuffle(); // 洗牌
        Poker.Card c1 = poker.deal(0); // 发第一张牌
        // 对于非静态内部类 Card
        // 只有通过其外部类 Poker 对象才能创建 Card 对象
        Poker.Card c2 = poker.new Card("红心", 1); // 自己创建一张牌
        System.out.println(c1); // 洗牌后的第一张
        System.out.println(c2); // 打印: 红心 A
    }
}
```

100、Java 中会存在内存泄漏吗，请简单描述。

理论 上 Java 因为 有垃 圾回 收机 制（ GC ）不 会存 在内 存泄 露问 题（ 这也 是 Java 被广泛 使用 于服 务器 端编 程的 一个 重要 原因 ）； 然而 在实 际开 发中 ，可 能会 存在 无用但 可达 的对 象，这些 对象 不能 被 GC 回收 ，因此 也会 导致 内存 泄露 的发 生 。

例 如Hibernate 的 Session（ 一级 缓存 ）中的 对象 属于 持久 态，垃圾 回收 器是 不会 回收这些 对象 的，然而 这些 对象 中可 能存 在无 用的 垃圾 对象 ，如果 不及 时关 闭（ close ）或清 空（ flush ）一 级缓 存就 可能 导致 内存 泄露 。下 面例 子中 的代 码也 会导 致内 存泄露

```
import java.util.Arrays;
import java.util.EmptyStackException;
public class MyStack<T> {
```

```
    private T[] elements;
    private int size = 0;
    private static final int INIT_CAPACITY = 16;
    public MyStack() {
        elements = (T[]) new Object[INIT_CAPACITY];
    }
    public void push(T elem) {
        ensureCapacity();
        elements[size++] = elem;
    }
    public T pop() {
        if(size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }
    private void ensureCapacity() {
        if(elements.length == size) {
            elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }
}
```

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 方法却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

101、抽象的（abstract）方法是否可同时是静态的（static）,是否可同时是本地方法（native），是否可同时被 synchronized修饰？

都不能。抽象方法需要子类重写，而静态的方法是无法被重写的，因此二者是矛盾的。本地方法是由本地代码（如 C 代码）实现的方法，而抽象方法是没有实现的，也是矛盾的。synchronized 和方法的实现细节有关，抽象方法不涉及实现细节，因此也是相互矛盾的。

102、是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，在调用静态方法时可能对象并没有被初始化。

103、如何实现对象克隆？

有两种方式： 1). 实现 Cloneable 接口并重写 Object 类中的 clone()方法； 2). 实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真 正的深度克隆，代码如下。

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
public class MyUtil {
    private MyUtil() {
        throw new AssertionError();
    }
    @SuppressWarnings("unchecked")
    public static <T extends Serializable> T clone(T obj) throws
    Exception {
        ByteArrayOutputStream bout = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(bout);
        oos.writeObject(obj);
        ByteArrayInputStream bin = new
        ByteArrayInputStream(bout.toByteArray());
        ObjectInputStream ois = new ObjectInputStream(bin);
        return (T) ois.readObject();
// 说明：调用 ByteArrayInputStream 或 ByteArrayOutputStream对象的 close 方法没有任何
意义
// 这两个基于内存的流只要垃圾回收器清理对象就能够释放资源，这一点不同于对外部资源（如文件流）的
释放
    }
}
```

测试代码：

```
import java.io.Serializable;
class Person implements Serializable {
    private static final long serialVersionUID = -9102017020286042305L;
    private String name; // 姓名
    private int age; // 年龄
    private Car car; // 座驾
    public Person(String name, int age, Car car) {
        this.name = name;
        this.age = age;
        this.car = car;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public Car getCar() {
        return car;
    }
    public void setCar(Car car) {
```

```
        this.car = car;
    }
    @Override
    public String toString() {
        return "Person [name=" + name + ", age=" + age + ", car=" +
        car + "]";
    }
}
class Car implements Serializable {
    private static final long serialVersionUID = -5713945027627603702L;
    private String brand; // 品牌
    private int maxSpeed; // 最高时速
    public Car(String brand, int maxSpeed) {
        this.brand = brand;
        this.maxSpeed = maxSpeed;
    }
    public String getBrand() {
        return brand;
    }
    public void setBrand(String brand) {
        this.brand = brand;
    }
    public int getMaxSpeed() {
        return maxSpeed;
    }
    public void setMaxSpeed(int maxSpeed) {
        this.maxSpeed = maxSpeed;
    }
    @Override
    public String toString() {
        return "Car [brand=" + brand + ", maxSpeed=" + maxSpeed +
        "]";
    }
}
class CloneTest {
    public static void main(String[] args) {
        try {
                Person p1 = new Person("Hao LUO", 33, new Car("Benz",
                    300));
                Person p2 = MyUtil.clone(p1); // 深度克隆
                p2.getCar().setBrand("BYD");
                // 修改克隆的 Person 对象 p2 关联的汽车对象的品牌属性
                // 原来的 Person 对象 p1 关联的汽车不会受到任何影响
                // 因为在克隆 Person 对象时其关联的汽车对象也被克隆了
                System.out.println(p1);
            } catch (Exception e) {
                e.printStackTrace();
            }
    }
}
```

注意：基于 序列 化和 反序 列化 实现 的克 隆不 仅仅 是深 度克 隆， 更重 要的 是通 过泛型限 定， 可以 检查 出要 克隆 的对 象是 否支 持序 列化 ，这 项检 查是 编译 器完 成的 ，不是 在运 行时 抛出 异常 ，这种 是方 案明 显优 于使 用 Object 类的 clone 方法 克隆 对象。 让问 题在 编译 的时 候暴 露出 来总 是好 过把 问题 留到 运行 时。

104、接口是否可继承（extends）接口？抽象类是否可实现（implements）接口？抽象类是否可继承具体类（concreteclass）？

接口可以继承接口，而且支持多重继承。抽象类可以实现(implements)接口，抽象类可继承具体类也可以继承抽象类。

105、一个”.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？

可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

106、Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

可以继承其他类或实现其他接口，在 Swing 编程和 Android 开发中常用此方式来实现事件监听和回调。

107、内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员。

108、Java 中的 final 关键字有哪些用法？

(1)修饰类：表示该类不能被继承；

(2)修饰方法：表示方法不能被重写；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。