1. JDK是什么？JRE是什么？

答：JDK：java开发工具包。JRE：java运行时环境。

1. 什么是java的平台无关性？

答：Java源文件被编译成字节码的形式，无论在什么系统环境下，只要有java虚 拟机就能运行这个字节码文件。也就是一处编写，处处运行。这就是java的跨平台性。

1. 在一台电脑上配置java环境，path起什么作用？如何配置？

答：a）path的作用是在DOS环境下，能在任意位置使用JDK目录中bin文件夹中的 可执行程序，来编译执行java程序。 b) 在环境变量中找到path变量，把bin文件夹的绝对路径加上即可。

1. 什么样的标识符是合法的？

答： a) 由字母、数字、\_和$组成，长度不限。其中字母可以是大写或小写的英文字母，数 字为0到9。 b) 标识符的第一个字符不能是数字。 c) 标识符区分大小写。 d) 标识符不能包含空格。

1. Java有几种基本数据类型？

答：byte，short，int，long，char，boolean，float，double

1. 什么是隐式类型转换？什么是显示类型转换？

答：a) 当将占位数少的类型赋值给占位数多的类型时，Java自动使用隐式类型转换。 b) 当把在级别高的变量的值赋给级别底变量时，必须使用显示类型转换运算。

1. &&，&区别。||，|区别？

答：&&和||是短路与，短路或，当左边的表达式能判断当前结果，则不判断右边的表达式。

1. break，continue区别？

答：break结束最近的一个循环，continue结束当次循环，进入下次循环。

1. 类的命名规则是什么？

答： a) 如果类名使用拉丁字母，那么名字的首写字母使用大写字母。 b) 类名最好见名得意，当类名由几个单词复合而成时，每个单词的首写字母使用大写。

1. 类体的内容由两部分构成，哪两部分？

答： a) 一部分是变量的定义，用来刻画属性。 b) 另一部分是方法的定义，用来刻画功能。

1. 解释什么是类的成员变量，局部变量，实例成员变量，类成员变量？

答： a) 变量定义部分所定义的变量被称为类的成员变量。 b) 在方法体中定义的变量和方法的参数被称为局部变量。 c) 成员变量又分为实例成员变量和类成员变量（static修饰）。

1. 简述this关键字用法。

答：this关键字使用在实例方法中，代表调用该方法的当前对象。

1. 如何确定方法的返回类型？

答：方法返回的值的类型就是方法的返回类型，如果无返回值，则返回类型为void。

1. 返回值为void的方法，可否有return？

答：可以。但return后没有任何值。

1. 解释什么是类方法，什么是实例方法？

答：static修饰的方法是类方法，无static修饰的方法是实例方法。

1. 简述方法和变量的命名规则？

答：首写字母使用小写，如果由多个单词组成，从第2个单词开始首字母使用大写。

1. 什么是方法重载？

答：方法重载是指一个类中可以有多个方法具有相同的名字，但这些方法的参数必须不 同，即或者是参数的个数不同，或者是参数的类型不同。

1. 什么是构造方法？

答：构造方法是一种特殊方法，它的名字必须与它所在的类的名字完全相同，并且不返 回任何数据类型。

1. 如何创建一个对象？

答：使用new 运算符和类的构造方法为对象分配内存，如果类中没有构造方法，系统 会调用默认的构造方法。

1. 系统什么情况下会为类提供构造方法，提供什么样的构造方法？

答：如果类中没有构造方法，系统会提供一个默认的构造方法，默认的构造方法是无参 的。

1. 对象如何调用自己的成员变量和方法？

答：使用运算符”.”来调用自己的成员变量和方法。

1. 为什么可以直接用类名来访问类成员变量和类方法？

答：因为类被加载到虚拟机类的成员变量就被分配内存，类方法被分配入口地址，所以 不用创建对象，可以直接通过类名调用。

1. 类变量有什么特点？

答：一个类的所有对象共享同一个类变量。

1. 类方法有什么特点？

答： a) 类方法只能调用类变量和类方法。（同一类中） b) 其他类可以通过类名直接调用。

1. package关键字有什么作用，使用中注意什么问题？

答：package指定一个类所在的包，该语句为源代码第一行。

1. import关键字有什么作用？

答： 引入程序中所用到的类。

1. 类有几种访问权限？变量和方法有几种访问权限？分别是什么？

答： a) 类有两种访问权限：public，default。 b) 方法和变量有四种访问权限：public，protected，default，private。

1. 简述java的访问权限。

答： a) public：公有的，任何类都可以访问。 b) protected：受保护的，同一个包的类可以访问。不同包的子类可以访问。 c) default：同一个包的类可以访问。 d) private：私有的，在同一个类中才能访问。

1. 子类能继承父类的哪些变量和方法？

答： a) 如果子类和父类在同一个包中，那么，子类自然地继承了其父类中不是private 的 成员变量作为自己的成员变量，并且也自然地继承了父类中不是private 的方法作为自己的方法。 b) 如果子类和父类不在同一个包中，那么，子类继承了父类的protected，public 成员变量做为子类的成员变量，并且继承了父类的protected，public 方法为子类的方法。

1. 子类重写父类的方法，可否降低访问权限？

答：不可以降低。

1. final关键字可以用来修饰什么？分别起什么作用？

答：a) final可以修饰类，这样的类不能被继承。 b) final可以修饰方法，这样的方法不能被重写。 c) final可以修饰变量，这样的变量的值不能被修改，是常量。

1. 简述super关键字作用？

答： a) 使用super调用父类的构造方法。 b) 使用super操作被隐藏的成员变量和方法。

1. 简述什么是对象上转型？

答：假设，A 类是B 类的父类，当我们用子类创建一个对象，并把这个对象的引用放 到父类的对象中时，我们称这个父类对象是子类对象的上转型对象。

1. 上转型对象可以操作什么？不可以操作什么？

答： a) 上转对象不能操作子类新增的成员变量，失掉了这部分属性，不能使用子类新增的 方法，失掉了一些功能 。 b) 上转型对象可以操作子类继承或重写的成员变量，也可以使用子类继承的或重写的 方法。

1. 什么是抽象类？什么是抽象方法？有什么特点？

答： a) 用关键字abstract修饰类称为抽象类，abstract类不能用new运算创建对象，必须 产生其子类，由子类创建对象。 b) 用关键字abstract修饰类称为抽象方法，abstract方法，只允许声明，而不允许实 现。

1. 一个类声明实现一个接口，那么这个类需要做什么工作？

答：实现接口里所有的方法，并且这些方法的访问权限必须是public。

1. 简述什么是数组？

答：数组是相同类型的数据按顺序组成的一种复合数据类型。通过数组名加数组下标， 来使用数组中的数据。下标从 0 开始排序。

1. 创建数组是否需要指定数组长度，如何求数组长度？

答：必须指定数组长度，数组调用。length来获取数组长度。

1. char数组和byte数组如何转化成字符串？如何求字符串长度？

答：利用String类本身的构造方法，String (char a[])，String (byte a[])，””.length()。

1. equals equalsIgnoreCase startsWith endsWith

答：a) 字符串比较 。 b) 忽略大小写比较。 c) 判断是否以指定字符串开头、结尾。

1. regionMatches

答：判断一个字符串的字串是否和另一个字符串的字串匹配。

1. compareTo compareToIgnoreCase indexOf

答：a) 按字典顺序比较字符串。 b) 忽略大小写的按字典顺序比较字符串。 c) 字符

串检索。

1. substring replace trim()

答： a) 字符串截取 b) 字符串替换 c) 去掉字符串前后空格。

1. 字符串如何转化为int型，double型？

答：a)Integer.parseInt(“1”); b) Double.parseDouble(“25.45”);

1. 如何判断一个字符是不是数字，是不是大写？

答： a) Character.isDigit(‘a’) b) Character.isUpperCase(‘U’)

1. 已知一个java.util.Date对象，如何格式化成如下格式 YYYY-MM-dd hh:mm:ss

答：SimpleDateFormat formate = new SimpleDateFormat(""yyyy-MM-dd HH:mm:ss""); formate.format(new Date())); "

1. 如何生成一个0-100的随即整数？

答：(int)(Math.random()\*100)

1. 简述java异常处理的机制？

答：当所调用的方法出现异常时，调用者可以捕获异常使之得到处理；也可以回避异常。

1. 简述如下关键字用途try、catch、throw、throws、finally

答： a) try：保护代码，如果try中某行代码出现异常，则try中代码不再继续执行。 b) catch：捕获异常，当try中出现异常，则catch负责捕获异常，并处理。 c) throw：抛出异常。 d) throws：声明异常。 e) finally：无论try中是否出现异常，finally一定执行。

1. LinkedList和ArrayList区别？

答： a) LinkedList底层通过链式存储结构实现。 b) ArrayList底层通过数组实现。

1. 什么是I/O流，有什么作用？

答：指数据输入输出的流， I/O 流提供一条通道程序，可以使用这条通道把源中的字 节序列送给目的地。

1. 如何查看一个文件的大小，绝对路径，是否可读？

答：a)File file = new File(“e://a.txt”); b) file.length(); c) file.getAbsolutePath(); d) file.canRead();

1. 已知如下代码 File f = new File(""myfile.dat""); FileInputStream istream = new

FileInputStream(f); 如何从流中读取数据。

答：byte[] buff = new byte[100]; b) istream.read(buff); "

1. 构造方法能否被重写？为什么？

答：不能，因为构造方法不能被继承，所以不能重写。

1. java关键字区分大小写吗？

答：java关键字一律小写。所以无所谓区分大小写，大写的不是关键字。

1. java采用什么字符集？该字符集有多少字符

答：Java 使用 unicode 字符集,所以常量共有 65535 个

1. 列举算术运算符？

答： +，-，\*，/，%

1. 算术混合运算结果精度如何确定？

答：Java按照运算符两边的操作元的最高精度保留结果的精度。

1. &是位运算符，与运算的规则是什么？

答：全1则1，否则为0

1. | 是位运算符，或运算的规则是什么？

答：全0则0，否则是1

1. ^是位运算符，异或运算的规则是什么？

答：相同0，不同1

1. ~是位运算符，非运算的规则是什么？

答：遇1则0，遇0则1

1. if语句后边有个括号，该括号里表达式为什么类型？

答：boolean类型

1. switch语句后括号里的表达式是什么类型？case后类型如何确定？

答：a) char，int，byte，short，枚举类型。 b) case后面是一个常量，该常量类型由switch

后括号内表达式来确定。

1. 使用final关键字修饰一个变量时，是引用不能变，还是引用的对象不能变？

答：final关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象变量中个内容是可以改变的。例如：final StringBuffer sb=new StringBuffer();

1. 在创建对象时，必须先声明对象，为对象分配内存空间，为对对象初始化，然后才

能使用对象 ？

答：错误

1. 所谓抽象类就是包含有抽象方法的类？

答：错误。抽象类，就是不能自己单独实例化的类。里面的方法不要求一定有抽象方法

1. 当一个方法在运行过程中产生一个异常，则这个方法会终止，但是整个程序要终止

运行?

答：错误 。整个应用程序正常运行

1. Java语言不含有指针与析构方法，所以java程序只有在结束运行时才能释放所占用

的内存？

答：错误：垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收

1. 重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类

中多态性的一种表现？

答：正确

1. 在java语言中,每个ASCLL码英文字符占用8个二进制位,而每个中文汉字字符则占

用16个二进制位。

答：错误 java中的字符是Unicode字符集 都是16位。

1. Java方法调用时参数都是按值传递的，从方法退出时，不会影响调用方的参数值

答：正确

1. 类A和类B位于同一个包中，则除了私有成员，类A可以访问类B的所有其他成

员？

答：正确

1. Abstract的类，是绝对不能被new的，也就是不会出现 new AbstractClass 这种情

况？

答：错误 是可以new

匿名类可以直接来new 一个抽象类，只要实现了里面的抽象方法

1. 在异常处理中，若try中的代码可能产生多种异常则可以对应多个catch语句，若

catch中的参数类型有父类子类关系，此时应该将父类放在后面，子类放在前面。

答：正确

1. 在Java中，异常Exception是指程序在编译和运行进出现的错误

答：错误 Exception 是运行时的异常。

1. Java类中不能存在同名的两个成员函数

答：错误：参数不同即可，用来实现oveloaded

1. swtich是否能作用在enum上？

答：可以

1. java代码安全性是类加载器通过分离本机文件系统的类和从网络导入的类增加安

全性？

答：正确

1. Java是一种完全面象对象的程序设计语言，并且是一种绝对安全的语言 ？

答：错误。 相对安全的语言

1. 如果方法没有返回值，则要一定要指定为void?

答：错误 。构造函数无返回类型

1. 一个java源程序编译通过后，得到的结果文件数也只有一个?

答：错误 。 一个文件可以包含多个class,此时编译时会产生多个文件

1. 如果一个对象equals另一个对象,那么他们的hashcode一定相同？

答：不一定，equals的实现不一定和hashcode有关系

1. 面向对象的特征有哪些方面?

答：面向对象的特征主要有以下几个方面：1)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。2)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）。3)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）。4)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）。

1. String 是最基本的数据类型吗?

答：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）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

1. 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;。

1. short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

答：short s1 = 1; s1 = s1 + 1;有错，s1是short型，s1+1是int型,不能显式转化为short型。可修改为s1 =(short)(s1 + 1) 。short s1 = 1; s1 += 1正确。因为s1+= 1;相当于s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

1. Java 有没有goto?

答：goto 是Java中的保留字，在目前版本的Java中没有使用。（根据James Gosling（Java之父）编写的《The Java Programming Language》一书的附录中给出了一个Java关键字列表，其中有goto和const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉C语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

1. int 和Integer 有什么区别?

答：Java是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java为每一个基本数据类型都引入了对应的包装类型（wrapper class），int的包装类就是Integer，从JDK 1.5开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：原始类型: boolean，char，byte，short，int，long，float，double包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

[java] view plain copy 在CODE上查看代码片派生到我的代码片

package com.lovo;

public static void main(String[] args) {

Integer a = new Integer(3);

Integer b = 3; // 将3自动装箱成Integer类型

int c = 3;

System.out.println(a == b); // false 两个引用没有引用同一对象

System.out.println(a == c); // true a自动拆箱成int类型再和c比较

}

}

补充：最近还遇到一个面试题，也是和自动装箱和拆箱相关的，代码如下所示：

[java] view plain copy 在CODE上查看代码片派生到我的代码片

public class Test03 {

public static void main(String[] args) {

Integer f1 = 100, f2 = 100, f3 = 150, f4 = 150;

System.out.println(f1 == f2);

System.out.println(f3 == f4);

}

}

如果不明就里很容易认为两个输出要么都是true要么都是false。首先需要注意的是f1、

f2、f3、f4四个变量都是Integer对象，所以下面的==运算比较的不是值而是引用。装

箱的本质是什么呢？当我们给一个Integer对象赋一个int值的时候，会调用Integer类

的静态方法valueOf，如果看看valueOf的源代码就知道发生了什么。

public static Integer valueOf(int i) {

if (i >= IntegerCache.low && i <= IntegerCache.high)

Return IntergerCache.cache[i + (-IntegerCache.low)];

return new Integer(i);

}

IntegerCache是Integer的内部类，其代码如下所示：

private static class IntegerCache {

static final int low = -128;

static final int high;

static final Integer cache[];

static {

// high value may be configured by property

int h = 127;

String integerCacheHighPropValue =

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");

if (integerCacheHighPropValue != null) {

try {

int i = parseInt(integerCacheHighPropValue);

i = Math.max(i, 127);

// Maximum array size is Integer.MAX\_VALUE

h = Math.min(i, Integer.MAX\_VALUE - (-low) -1);

} catch( NumberFormatException nfe) {

// If the property cannot be parsed into an int, ignore it.

}

}

high = h;

cache = new Integer[(high - low) + 1];

int j = low;

for(int k = 0; k < cache.length; k++)

cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)

assert IntegerCache.high >= 127;

}

private IntegerCache() {}

}

简单的说，如果字面量的值在-128到127之间，那么不会new新的Integer对象，而是

直接引用常量池中的Integer对象，所以上面的面试题中f1==f2的结果是true，而f3==f4

的结果是false。越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的

功力。

1. &和&&的区别？

答：&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是true整个表达式的值才是true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是null而且不是空字符串，应当写为：username != null &&!username.equals(“”)，二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的equals比较，否则会产生NullPointerException异常。注意：逻辑或运算符（|）和短路或运算符（||）的差别也是如此。

补充：如果你熟悉JavaScript，那你可能更能感受到短路运算的强大，想成为JavaScript的高手就先从玩转短路运算开始吧。

1. 解释内存中的栈（stack）、堆(heap)和静态存储区的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过new关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的100、“hello”和常量都是放在静态存储区中。栈空间操作最快但是也很小，通常大量的对象都是放在堆空间，整个内存包括硬盘上的虚拟内存都可以被当成堆空间来使用。

String str = new String(“hello”);

上面的语句中str放在栈上，用new创建出来的字符串对象放在堆上，而“hello”这个字面量放在静态存储区。

补充：较新版本的Java中使用了一项叫“逃逸分析“的技术，可以将一些局部对象放在栈上以提升对象的操作性能。

1. Math.round(11.5) 等于多少? Math.round(-11.5)等于多少?

答：Math.round(11.5)的返回值是12，Math.round(-11.5)的返回值是-11。四舍五入的原理是在参数上加0.5然后进行下取整。

1. swtich 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上?

答：早期的JDK中，switch（expr）中，expr可以是byte、short、char、int。从1.5版开始，Java中引入了枚举类型（enum），expr也可以是枚举，从JDK 1.7版开始，还可以是字符串（String）。长整型（long）是不可以的。

1. 用最有效率的方法计算2乘以8?

答： 2 << 3（左移3位相当于乘以2的3次方，右移3位相当于除以2的3次方）。

补充：我们为编写的类重写hashCode方法时，可能会看到如下所示的代码，其实我们不太理解为什么要使用这样的乘法运算来产生哈希码（散列码），而且为什么这个数是个素数，为什么通常选择31这个数？前两个问题的答案你可以自己百度一下，选择31是因为可以用移位和减法运算来代替乘法，从而得到更好的性能。说到这里你可能已经想到了：31 \* num <==> (num << 5) - num，左移5位相当于乘以2的5次方（32）再减去自身就相当于乘以31。现在的VM都能自动完成这个优化。

package com.loonstudio;

public class PhoneNumber {

private int areaCode;

private String prefix;

private String lineNumber;

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + areaCode;

result = prime \* result + ((lineNumber == null) ? 0 : lineNumber.hashCode());

result = prime \* result + ((prefix == null) ? 0 : prefix.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

PhoneNumber other = (PhoneNumber) obj;

if (areaCode != other.areaCode)

return false;

if (lineNumber == null) {

if (other.lineNumber != null)

return false;

} else if (!lineNumber.equals(other.lineNumber))

return false;

if (prefix == null) {

if (other.prefix != null)

return false;

} else if (!prefix.equals(other.prefix))

return false;

return true;

}

}

1. 数组有没有length()方法?String 有没有length()方法？

答：数组没有length()方法，有length 的属性。String 有length()方法。JavaScript中，获得字符串的长度是通过length属性得到的，这一点容易和Java混淆。

1. 在Java 中，如何跳出当前的多重嵌套循环？

答：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）

1. 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。

1. 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？

答：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）。

补充：关于equals和hashCode方法，很多Java程序都知道，但很多人也就是仅仅知道而已，在Joshua Bloch的大作《Effective Java》（很多软件公司，《Effective Java》、《Java编程思想》以及《重构：改善既有代码质量》是Java程序员必看书籍，如果你还没看过，那就赶紧去亚马逊买一本吧）中是这样介绍equals方法的：首先equals方法必须满足自反性（x.equals(x)必须返回true）、对称性（x.equals(y)返回true时，y.equals(x)也必须返回true）、传递性（x.equals(y)和y.equals(z)都返回true时，x.equals(z)也必须返回true）和一致性（当x和y引用的对象信息没有被修改时，多次调用x.equals(y)应该得到同样的返回值），而且对于任何非null值的引用x，x.equals(null)必须返回false。实现高质量的equals方法的诀窍包括：1. 使用==操作符检查“参数是否为这个对象的引用”；2. 使用instanceof操作符检查“参数是否为正确的类型”；3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；4. 编写完equals方法后，问自己它是否满足对称性、传递性、一致性；5. 重写equals时总是要重写hashCode；6. 不要将equals方法参数中的Object对象替换为其他的类型，在重写时不要忘掉@Override注解。

1. 是否可以继承String 类?

答：String 类是final类，不可以被继承。

补充：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（HAS-A）而不是继承（IS-A）。

1. 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?

答：是值传递。Java 编程语言只有值传递参数。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，但对象的引用是永远不会改变的。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。

补充：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。

1. String 和StringBuilder、StringBuffer 的区别?

答：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和StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是JDK 1.5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略高。

补充1：有一个面试题问：有没有哪种情况用+做字符串连接比调用StringBuffer / StringBuilder对象的append方法性能更好？如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于StringBuffer / StringBuilder的append方法的。

补充2：下面也是一个面试题，问程序的输出，看看自己能不能说出正确答案。

package com.lovo;

public class StringEqualTest {

public static void main(String[] args) {

String a = ""Programming""; "

String b = new String(""Programming""); "

String c = ""Program"" + ""ming"";

System.out.println(a == b);

System.out.println(a == c);

System.out.println(a.equals(b));

System.out.println(a.equals(c));

System.out.println(a.intern() == b.intern());

}

}ftttt

1. 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?

答：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。

补充：华为的面试题中曾经问过这样一个问题：为什么不能根据返回类型来区分重载，说出你的答案吧！

1. 描述一下JVM 加载class文件的原理机制?

答：JVM 中类的装载是由类加载器（ClassLoader） 和它的子类来实现的，Java中的类加载器是一个重要的Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

补充：1）.由于Java的跨平台性，经过编译的Java源程序并不是一个可执行程序，而是一个或多个类文件。当Java程序需要使用某个类时，JVM会确保这个类已经被加载、连接(验证、准备和解析)和初始化。类的加载是指把类的.class文件中的数据读入到内存中，通常是创建一个字节数组读入.class文件，然后产生与所加载类对应的Class对象。加载完成后，Class对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备(为静态变量分配内存并设置默认的初始值)和解析(将符号引用替换为直接引用)三个步骤。最后JVM对类进行初始化，包括：1如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2如果类中存在初始化语句，就依次执行这些初始化语句。

2）.类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类）。从JDK 1.2开始，类加载过程采取了父亲委托机制(PDM)。PDM更好的保证了Java平台的安全性，在该机制中，JVM自带的Bootstrap是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM不会向Java程序提供对Bootstrap的引用。下面是关于几个类加载器的说明：

a)Bootstrap：一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；

b)Extension：从java.ext.dirs系统属性所指定的目录中加载类库，它的父加载器是Bootstrap；

c)System：又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量classpath或者系统属性java.class.path所指定的目录中记载类，是用户自定义加载器的默认父加载器。

1. char 型变量中能不能存贮一个中文汉字?为什么?

答：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（16bit），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。

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

答：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。

package com.lovo;

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：

package com.lovo;

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

}

}

1. Java 中会存在内存泄漏吗，请简单描述。

答：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回收也会发生内存泄露。一个例子就是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。下面的例子也展示了Java中发生内存泄露的情况：

package com.lovo;

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。

1. 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?

答：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。

1. 静态变量和实例变量的区别？

答：静态变量是被static修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。在Java开发中，上下文类和工具类中通常会有大量的静态成员。

1. 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？

答：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，因此在调用静态方法时可能对象并没有被初始化。

1. 如何实现对象克隆？

答：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。

package com.lovo;

import java.io.ByteArrayInputStream;

import java.io.ByteArrayOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

public class MyUtil {

private MyUtil() {

throw new AssertionError();

}

public static <T> 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

}

}

下面是测试代码：

package com.lovo;

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 + ""]""; "

}

}

/\*\*

\* 小汽车类

\* @author 骆昊

\*

\*/

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。

1. Anonymous Inner Class（匿名内部类）是否可以extends（继承）其它类，是否可

以implements（实现）interface（接口）？

答：匿名的内部类是没有名字的内部类。不能extends(继承) 其它类，但一个内部类可

以作为一个接口，由另一个内部类实现。

1. HashMap和Hashtable的区别。

答：都属于Map接口的类，实现了将惟一键映射到特定的值上。 HashMap 类没有分

类或者排序。它允许一个 null 键和多个 null 值。Hashtable 类似于 HashMap，但是不

允许 null 键和 null 值。它也比 HashMap 慢，因为它是同步的。

1. Java中异常处理机制，事件机制？

答：Java通过面向对象的方法来处理例外。在一个方法的运行过程中，如果发生了例外，则这个方法生成代表该例外的一个对象，并把它交给运行时系统，运行时系统寻找相应的代码来处理这一例外。我们把生成例外对象并把它提交给运行时系统的过程称为抛弃(throw)一个例外。运行时系统在方法的调用栈中查找，从生成例外的方法开始进行回朔，直到找到包含相应例外处理的方法为止，这一个过程称为捕获(catch)一个例外。类Throwable有两个直接子类：Error和Exception。Error类对象（如动态连接错误等），由Java虚拟机生成并抛弃（通常，Java程序不对这类例外进行处理）；Exception类对象是Java程序处理或抛弃的对象。它有各种不同的子类分别对应于不同类型的例外。其中类RuntimeException代表运行时由Java虚拟机生成的例外，如算术运算例外ArithmeticException(由除0错等导致)、数组越界例外ArrayIndexOutOfBoundsException等；其它则为非运行时例外，如输入输出例外IOException等。Java编译器要求Java程序必须捕获或声明所有的非运行时例外，但对运行时例外可以不做处理。throws总是出现在一个函数头中，用来标明该成员函数可能抛出的各种异常。对大多数Exception子类来说，Java 编译器会强迫你声明在一个成员函数中抛出的异常的类型。如果异常的类型是Error或 RuntimeException， 或它们的子类，这个规则不起作用， 因为这在程序的正常部分中是不期待出现的。 如果你想明确地抛出一个RuntimeException，你必须用throws语句来声明它的类型。

1. String s=new String(“xyz”);创建了几个字符串对象？

答：两个对象，一个是静态存储区的""xyz"",一个是用new创建在堆上的对象。"

1. 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（concrete class）?

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承具体类，但前提是具体类必须有明确的构造函数。

1. 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？

答：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。

1. 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？

答：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。

1. Java 中的final关键字有哪些用法？

答：(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）。

1. 指出下面程序的运行结果:

class A{

static{

System.out.print(""1"");

}

public A(){

System.out.print(""2"");

}

}

class B extends A{

static{

System.out.print(""a"");

}

public B(){

System.out.print(""b"");

}

}

public class Hello{

public static void main(String[] args){

A ab = new B();

ab = new B();

}

}

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

1. 数据类型之间的转换:

1)如何将字符串转换为基本数据类型？

2)如何将基本数据类型转换为字符串？

答：1)调用基本数据类型对应的包装类中的方法parseXXX(String)或valueOf(String)即可返回相应基本类型；

2)一种方法是将基本数据类型与空字符串（””）连接（+）即可获得其所对应的字符串；另一种方法是调用String 类中的valueOf(…)方法返回相应字符串

1. 如何实现字符串的反转及替换？

答：方法很多，可以自己写实现也可以使用String或StringBuffer / StringBuilder中的方法。有一道很常见的面试题是用递归实现字符串反转，代码如下所示：

public static String reverse(String originStr) {

if(originStr == null || originStr.length() <= 1)

return originStr;

return reverse(originStr.substring(1)) + originStr.charAt(0);

}

1. 怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串？

答：代码如下所示:

"String s1 = ""你好"";"

"String s2 = newString(s1.getBytes(""GB2312""), ""ISO-8859-1"");"

1. 日期和时间：

1)如何取得年月日、小时分钟秒？

2)如何取得从1970年1月1日0时0分0秒到现在的毫秒数？

3)如何取得某月的最后一天？

4)如何格式化日期？

答：操作方法如下所示：

1)创建java.util.Calendar 实例，调用其get()方法传入不同的参数即可获得参数所对应的值

2)以下方法均可获得该毫秒数:

Calendar.getInstance().getTimeInMillis();

System.currentTimeMillis();

3)示例代码如下:

[java] view plain copy 在CODE上查看代码片派生到我的代码片

Calendar time = Calendar.getInstance();

time.getActualMaximum(Calendar.DAY\_OF\_MONTH);

4)利用java.text.DataFormat 的子类（如SimpleDateFormat类）中的format(Date)方法可将日期格式化。

1. 打印昨天的当前时刻。

答：

public class YesterdayCurrent {

public static void main(String[] args){

Calendar cal = Calendar.getInstance();

cal.add(Calendar.DATE, -1);

System.out.println(cal.getTime());

}

}

1. 比较一下Java 和JavaSciprt。

答：JavaScript 与Java是两个公司开发的不同的两个产品。Java 是原Sun 公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而JavaScript是Netscape公司的产品，为了扩展Netscape浏览器的功能而开发的一种可以嵌入Web页面中运行的基于对象和事件驱动的解释性语言，它的前身是LiveScript；而Java 的前身是Oak语言。

下面对两种语言间的异同作如下比较：

1）基于对象和面向对象：Java是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语言。因而它本身提供了非常丰富的内部对象供设计人员使用；

2）解释和编译：Java 的源代码在执行之前，必须经过编译；JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行；

3）强类型变量和类型弱变量：Java采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript中变量声明，采用其弱类型。即变量在使用前不需作声明，而是解释器在运行时检查其数据类型；

4）代码格式不一样。

补充：上面列出的四点是原来所谓的标准答案中给出的。其实Java和JavaScript最重要的区别是一个是静态语言，一个是动态语言。目前的编程语言的发展趋势是函数式语言和动态语言。在Java中类（class）是一等公民，而JavaScript中函数（function）是一等公民。对于这种问题，在面试时还是用自己的语言回答会更加靠谱。

1. Error 和Exception 有什么区别?

答：Error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况；Exception 表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

补充：2005年摩托罗拉的面试中曾经问过这么一个问题“If a process reports a stack overflow run-time error, what’s the most possible cause?”，给了四个选项a. lack of memory; b. write on an invalid memory space; c. recursive function calling; d. array index out of boundary. Java程序在运行时也可能会遭遇StackOverflowError，这是一个错误无法恢复，只能重新修改代码了，这个面试题的答案是c。如果写了不能迅速收敛的递归，则很有可能引发栈溢出的错误，如下所示：

package com.lovo;

public class StackOverflowErrorTest {

public static void main(String[] args) {

main(null);

}

}

因此，用递归编写程序时一定要牢记两点：1. 递归公式；2. 收敛条件（什么时候就不再递归而是回溯了）。

1. try{}里有一个return语句，那么紧跟在这个try后的finally{}里的code会不会被执

行，什么时候被执行，在return前还是后?

答：会执行，在方法返回调用者前执行。Java允许在finally中改变返回值的做法是不好的，因为如果存在finally代码块，try中的return语句不会立马返回调用者，而是记录下返回值待finally代码块执行完毕之后再向调用者返回其值，然后如果在finally中修改了返回值，这会对程序造成很大的困扰，C#中就从语法上规定不能做这样的事。

1. Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally分别如

何使用？

答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在Java 中，每个异常都是一个对象，它是Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并进行处理。Java 的异常处理是通过5 个关键词来实现的：try、catch、throw、throws和finally。一般情况下是用try来执行一段程序，如果出现异常，系统会抛出（throw）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try用来指定一块预防所有“异常”的程序；catch 子句紧跟在try块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个try语句，在这个成员函数内部写另一个try语句保护其他代码。每当遇到一个try 语句，“异常”的框架就放到栈上面，直到所有的try语句都完成。如果下一级的try语句没有对某种“异常”进行处理，栈就会展开，直到遇到有处理这种“异常”的try 语句。

1. 运行时异常与受检异常有何异同？

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，神作《Effective Java》中对异常的使用给出了以下指导原则：

1）不要将异常处理用于正常的控制流（设计良好的API不应该强迫它的调用者为了正常的控制流而使用异常）

2）对可以恢复的情况使用受检异常，对编程错误使用运行时异常

3）避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）

4）优先使用标准的异常

5）每个方法抛出的异常都要有文档

6）保持异常的原子性

7）不要在catch中忽略掉捕获到的异常

1. 列出一些你常见的运行时异常？

答：ArithmeticException（算术异常）

ClassCastException （类转换异常）

IllegalArgumentException （非法参数异常）

IndexOutOfBoundsException （下表越界异常）

NullPointerException （空指针异常）

SecurityException （安全异常）

1. 声明和访问控制

答：声明，构建，初始化任何类型的数组

声明类，内部类，方法，成员变量，静态成员变量和方法变量，并会应用任何合法的修饰符（如public,final,static,abstract,等等）。能够明了这些修饰符单独和组合起来的含义，并且知道被修饰符修饰的任意对象在各种包相关联系下的影响。

1. 流程控制，断言和异常处理

答：1）能够正确使用if,switch语句并且在这些语句中能正确使用合法的参数类型。

2）能够正确使用所有带标签或不带标签的循环语句，能使用break,continue，能计算在循环中或循环后循环计数器的值。

3）能够正确使用异常和异常处理语句（try,catch,finally）。能正确声明掷出例外的方法，并知道怎样覆盖它。

4）知道在程序段的特定点出现的异常对程序的影响。即：异常可能是一个runtime exception,一个checked exception也可能是一个error。（这个程序段可能包括try,catch,finally并以任何可能的合法组合出现。）。

5）能正确应用断言，区分正确使用的断言和不正确的。

6）明白关于断言机制的正确说法。

1. 垃圾收集

答：1）明白垃圾收集机制确定性的行为。

2）能用程序显式的使一个对象能被垃圾收集器合法的收集。

3）知道在程序的哪一点垃圾收集器能合法地收集一个对象。

1. 语言基础

答：1）能正确构建包声明，import声明，类声明（包括内部类），接口声明，方法声明（包括用于开始一个类的执行的main方法），变量声明及其其它的一些说明符。

2）能够正确使用一些类，这些类要么实现了java.lang.Runnable这个接口，要么能正确实现在问题中构建的一些接口。

3）知道传入main函数的命令行参数的index value。

4）知道所有JAVA的keyword。注意：考试中不会出现要你区分keyword和各种常数这类深奥的问题。

5）明白如果没有显式地赋值的各种变量或者数组被使用会出现什么结果。

6）知道所有原始数据类型的取值范围，怎样声明一个String的字面值等等。

1. 操作与赋值

答：1）能知道当任何操作符（包括赋值操作符和intanceof操作符）应用于任何操作数（任何类型的类或访问能力或两者的任意组合）的结果。

2）知道String，Boolean和Object类使用equals(Object)方法后的结果。

知道当对已经知道值的变量进行&,|,&&,||操作时，哪些操作数被运算了，表达式最终的结果是怎样的。

3）知道Object和原始类型数据传入方法的不同方式，知道如何在这些方法中如何进行赋值或其它修改操作。

1. 覆盖，重载，运行时期类型及其面向对象

答：1）知道面向对象设计中封装的好处并能用程序实现紧密封装的类，能知道is a和has a的意义。

2）能正确使用覆盖和重载的方法，能正确调用父类或覆盖了的构建器，知道调用这些方法后的结果。

3）能实例化任何具体的一般顶层类和内部类。

1. 在java.lang包中的基础类

答：1）能够应用Math类中的abs,ceil,floor,max,min,random,round,sin,cos,tan,sqrt方法。

正确描述String类不可改变的意义。

2）当执行一段程序，中间包含有wrapper类的一个实例，知道它运行的前提条件运行3）结果。能用下面wrapper类（例如Integer,Double,等等）的方法来写程序：

doublevalue

floatvalue

intvalue

longvalue

parseXxx

getXxx

toString

toHexString

1. 集合类框架

答：1）知道如何在特定的条件下选择适合的集合类/接口。

2）区分正确和不正确对hashcode方法的实现。

1. 谈谈final, finally, finalize的区别。

答：1）final—修饰符（关键字）如果一个类被声明为final，意味着它不能再派生出新的子类，不能作为父类被继承。因此一个类不能既被声明为 abstract的，又被声明为final的。将变量或方法声明为final，可以保证它们在使用中不被改变。被声明为final的变量必须在声明时给定初值，而在以后的引用中只能读取，不可修改。被声明为final的方法也同样只能使用，不能重载

2）finally—再异常处理时提供 finally 块来执行任何清除操作。如果抛出一个异常，那么相匹配的 catch 子句就会执行，然后控制就会进入 finally 块（如果有的话）。

3）finalize—方法名。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的。它是在 Object 类中定义的，因此所有的类都继承了它。子类覆盖 finalize() 方法以整理系统资源或者执行其他清理工作。finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的。

1. Collection 和 Collections的区别。

答：Collections是个java.util下的类，它包含有各种有关集合操作的静态方法。

Collection是个java.util下的接口，它是各种集合结构的父接口.

1. Overload和Override的区别。Overloaded的方法是否可以改变返回值的类型?

答：方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded的方法是可以改变返回值的类型。

1. Set里的元素是不能重复的，那么用什么方法来区分重复与否呢? 是用==还是

equals()? 它们有何区别?

答：Set里的元素是不能重复的，那么用iterator()方法来区分重复与否。equals()是判读两个Set是否相等。

equals()和==方法决定引用值是否指向同一对象equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值。

1. List, Set, Map是否继承自Collection接口?

答：List，Set是

　　Map不是

1. abstract class和interface有什么区别?

答：声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类为。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。

接口（interface）是抽象类的变体。在接口中，所有方法都是抽象的。多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，没有一个有程序体。接口只可以定义static final成员变量。接口的实现与子类相似，除了该实现类不能从接口定义中继承行为。当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

1. abstract的method是否可同时是static,是否可同时是native，是否可同时是

synchronized?

答：都不能

1. try {}里有一个return语句，那么紧跟在这个try后的finally {}里的code会不会被

执行，什么时候被执行，在return前还是后?

答：会执行，在return前执行。

1. 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?

答：不对，有相同的hash code。

1. 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?

答：是值传递。Java 编程语言只由值传递参数。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。对象的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

1. 编程题: 写一个Singleton出来。

答：Singleton模式主要作用是保证在Java应用程序中，一个类Class只有一个实例存在。

一般Singleton模式通常有几种种形式:

第一种形式: 定义一个类，它的构造函数为private的，它有一个static的private的该类变量，在类初始化时实例话，通过一个public的getInstance方法获取对它的引用,继而调用其中的方法。

　　public class Singleton {

　　private Singleton(){}

　　//在自己内部定义自己一个实例，是不是很奇怪？

　　//注意这是private 只供内部调用

　　private static Singleton instance = new Singleton();

　　//这里提供了一个供外部访问本class的静态方法，可以直接访问

　　public static Singleton getInstance() {

　　return instance;

　　}

　　}

　　第二种形式:

　　public class Singleton {

private static Singleton instance = null;

private Singleton (){}

　　public static synchronized Singleton getInstance() {

　　//这个方法比上面有所改进，不用每次都进行生成对象，只是第一次

　　//使用时生成实例，提高了效率！

　　if (instance==null)

　　instance＝new Singleton();

　　return instance; }

　　}

　　其他形式:

　　定义一个类，它的构造函数为private的，所有方法为static的。

一般认为第一种形式要更加安全

1. Java的接口和C++的虚类的相同和不同处。

答：由于Java不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面

的方法或属性，现有的单继承机制就不能满足要求。与继承相比，接口有更高的灵活性，

因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的

方法和属性，并且接口里面的属性在默认状态下面都是public static,所有方法默认情况

下是public.一个类可以实现多个接口。

1. 垃圾回收的优点和原理。并考虑2种回收机制。

答："Java语言中一个显著的特点就是引入了垃圾回收机制，使c++程序员最头疼的内存

管理的问题迎刃而解，它使得Java程序员在编写程序的时候不再需要考虑内存管理。

由于有个垃圾回收机制，Java中的对象不再有"作用域"的概念，只有对象的引用才有"

作用域"。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收

器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的

或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个

对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃

圾回收。

1. 给我一个你最常见到的runtime exception。

答：ArithmeticException, ArrayStoreException, BufferOverflowException,

BufferUnderflowException, CannotRedoException, CannotUndoException,

ClassCastException, CMMException, ConcurrentModificationException,

DOMException, EmptyStackException, IllegalArgumentException,

IllegalMonitorStateException, IllegalPathStateException, IllegalStateException,

ImagingOpException, IndexOutOfBoundsException, MissingResourceException,

NegativeArraySizeException, NoSuchElementException, NullPointerException,

ProfileDataException, ProviderException, RasterFormatException, SecurityException,

SystemException, UndeclaredThrowableException, UnmodifiableSetException,

UnsupportedOperationException

1. 作用域public，private，protected，以及不写时的区别。

答：1、public：public表明该数据成员、成员函数是对所有用户开放的，所有用户都可

以直接进行调用

2、private：private表示私有，私有的意思就是除了class自己之外，任何人都不可以直接使用

3、protected：protected对于子女、朋友来说，就是public的，可以自由使用，没有任何限制，而对于其他的外部class，protected就变成private。

作用域 当前类 同一package 子孙类 其他package

public √ √ √ √

protected √ √ √ ×

default √ √ × ×

private √ × × ×

不写时默认为default

1. abstract的method是否可同时是static，是否可同时是native，是否可同时是

synchronized？

答：都不能

1. 描述一下你最常用的编程风格。

答： （1） 类名首字母应该大写。字段、方法以及对象（句柄）的首字母应小写。对于所有标识符，其中包含的所有单词都应紧靠在一起，而且大写中间单词的首字母。Java包（Package）属于一种特殊情况：它们全都是小写字母，即便中间的单词亦是如此。对于域名扩展名称，如com，org，net或者edu等，全部都应小写（这也是Java 1.1和Java 1.2的区别之一）。

（2） 为了常规用途而创建一个类时，请采取“经典形式”，并包含对下述元素的定义：equals()hashCode()toString()clone()（implement Cloneable）implement Serializable （3） 对于自己创建的每一个类，都考虑置入一个main()，其中包含了用于测试那个类的代码。为使用一个项目中的类，我们没必要删除测试代码。若进行了任何形式的改动，可方便地返回测试。这些代码也可作为如何使用类的一个示例使用。

（4） 应将方法设计成简要的、功能性单元，用它描述和实现一个不连续的类接口部分。理想情况下，方法应简明扼要。若长度很大，可考虑通过某种方式将其分割成较短的几个方法。这样做也便于类内代码的重复使用（有些时候，方法必须非常大，但它们仍应只做同样的一件事情）。

（5） 设计一个类时，请设身处地为客户程序员考虑一下（类的使用方法应该是非常明确的）。然后，再设身处地为管理代码的人考虑一下（预计有可能进行哪些形式的修改，想想用什么方法可把它们变得更简单）。

（6） 使类尽可能短小精悍，而且只解决一个特定的问题。下面是对类设计的一些建议：一个复杂的开关语句：考虑采用“多形”机制数量众多的方法涉及到类型差别极大的操作：考虑用几个类来分别实现许多成员变量在特征上有很大的差别：考虑使用几个类

（7） 让一切东西都尽可能地“私有”——private。可使库的某一部分“公共化”（一个方法、类或者一个字段等等），就永远不能把它拿出。若强行拿出，就可能破坏其他人现有的代码，使他们不得不重新编写和设计。若只公布自己必须公布的，就可放心大胆地改变其他任何东西。在多线程环境中，隐私是特别重要的一个因素——只有private字段才能在非同步使用的情况下受到保护。

（8） 谨惕“巨大对象综合症”。对一些习惯于顺序编程思维、且初涉OOP领域的新手，往往喜欢先写一个顺序执行的程序，再把它嵌入一个或两个巨大的对象里。根据编程原理，对象表达的应该是应用程序的概念，而非应用程序本身。

（9） 若不得已进行一些不太雅观的编程，至少应该把那些代码置于一个类的内部。 （10） 任何时候只要发现类与类之间结合得非常紧密，就需要考虑是否采用内部类，从而改善编码及维护工作（参见第14章14.1.2小节的“用内部类改进代码”）。 （11） 尽可能细致地加上注释，并用javadoc注释文档语法生成自己的程序文档。 （12） 避免使用“魔术数字”，这些数字很难与代码很好地配合。如以后需要修改它，无疑会成为一场噩梦，因为根本不知道“100”到底是指“数组大小”还是“其他全然不同的东西”。所以，我们应创建一个常数，并为其使用具有说服力的描述性名称，并在整个程序中都采用常数标识符。这样可使程序更易理解以及更易维护。

（13） 涉及构建器和异常的时候，通常希望重新丢弃在构建器中捕获的任何异常——如果它造成了那个对象的创建失败。这样一来，调用者就不会以为那个对象已正确地创建，从而盲目地继续。

（14） 当客户程序员用完对象以后，若你的类要求进行任何清除工作，可考虑将清除代码置于一个良好定义的方法里，采用类似于cleanup()这样的名字，明确表明自己的用途。除此以外，可在类内放置一个boolean（布尔）标记，指出对象是否已被清除。在类的finalize()方法里，请确定对象已被清除，并已丢弃了从RuntimeException继承的一个类（如果还没有的话），从而指出一个编程错误。在采取象这样的方案之前，请确定finalize()能够在自己的系统中工作（可能需要调用System.runFinalizersOnExit（true），从而确保这一行为）。

（15） 在一个特定的作用域内，若一个对象必须清除（非由垃圾收集机制处理），请采用下述方法：初始化对象；若成功，则立即进入一个含有finally从句的try块，开始清除工作。

（16） 若在初始化过程中需要覆盖（取消）finalize()，请记住调用super.finalize()（若Object属于我们的直接超类，则无此必要）。在对finalize()进行覆盖的过程中，对super.finalize()的调用应属于最后一个行动，而不应是第一个行动，这样可确保在需要基础类组件的时候它们依然有效。

（17） 创建大小固定的对象集合时，请将它们传输至一个数组（若准备从一个方法里返回这个集合，更应如此操作）。这样一来，我们就可享受到数组在编译期进行类型检查的好处。此外，为使用它们，数组的接收者也许并不需要将对象“造型”到数组里。

（18） 尽量使用interfaces，不要使用abstract类。若已知某样东西准备成为一个基础类，那么第一个选择应是将其变成一个interface（接口）。只有在不得不使用方法定义或者成员变量的时候，才需要将其变成一个abstract（抽象）类。接口主要描述了客户希望做什么事情，而一个类则致力于（或允许）具体的实施细节。

（19） 在构建器内部，只进行那些将对象设为正确状态所需的工作。尽可能地避免调用其他方法，因为那些方法可能被其他人覆盖或取消，从而在构建过程中产生不可预知的结果（参见第7章的详细说明）。

（20） 对象不应只是简单地容纳一些数据；它们的行为也应得到良好的定义。

（21） 在现成类的基础上创建新类时，请首先选择“新建”或“创作”。只有自己的设计要求必须继承时，才应考虑这方面的问题。若在本来允许新建的场合使用了继承，则整个设计会变得没有必要地复杂。

（22） 用继承及方法覆盖来表示行为间的差异，而用字段表示状态间的区别。一个非常极端的例子是通过对不同类的继承来表示颜色，这是绝对应该避免的：应直接使用一个“颜色”字段。

（23） 为避免编程时遇到麻烦，请保证在自己类路径指到的任何地方，每个名字都仅对应一个类。否则，编译器可能先找到同名的另一个类，并报告出错消息。若怀疑自己碰到了类路径问题，请试试在类路径的每一个起点，搜索一下同名的.class文件。

（24） 在Java 1.1 AWT中使用事件“适配器”时，特别容易碰到一个陷阱。若覆盖了某个适配器方法，同时拼写方法没有特别讲究，最后的结果就是新添加一个方法，而不是覆盖现成方法。然而，由于这样做是完全合法的，所以不会从编译器或运行期系统获得任何出错提示——只不过代码的工作就变得不正常了。

（25） 用合理的设计方案消除“伪功能”。也就是说，假若只需要创建类的一个对象，就不要提前限制自己使用应用程序，并加上一条“只生成其中一个”注释。请考虑将其封装成一个“独生子”的形式。若在主程序里有大量散乱的代码，用于创建自己的对象，请考虑采纳一种创造性的方案，将些代码封装起来。

（26） 警惕“分析瘫痪”。请记住，无论如何都要提前了解整个项目的状况，再去考察其中的细节。由于把握了全局，可快速认识自己未知的一些因素，防止在考察细节的时候陷入“死逻辑”中。

（27） 警惕“过早优化”。首先让它运行起来，再考虑变得更快——但只有在自己必须这样做、而且经证实在某部分代码中的确存在一个性能瓶颈的时候，才应进行优化。除非用专门的工具分析瓶颈，否则很有可能是在浪费自己的时间。性能提升的隐含代价是自己的代码变得难于理解，而且难于维护。

（28） 请记住，阅读代码的时间比写代码的时间多得多。思路清晰的设计可获得易于理解的程序，但注释、细致的解释以及一些示例往往具有不可估量的价值。无论对你自己，还是对后来的人，它们都是相当重要的。如对此仍有怀疑，那么请试想自己试图从联机Java文档里找出有用信息时碰到的挫折，这样或许能将你说服。

1. SQL问答题

表结构：

1） 表名：g\_cardapply

字段（字段名/类型/长度）：

g\_applyno varchar 8; //申请单号（关键字）

g\_applydate bigint 8; //申请日期

g\_state varchar 2; //申请状态

2） 表名：g\_cardapplydetail

字段（字段名/类型/长度）：

g\_applyno varchar 8; //申请单号（关键字）

g\_name varchar 30;//申请人姓名

g\_idcard varchar 18;//申请人身份证号

g\_state varchar 2;//申请状态

其中，两个表的关联字为申请单号。

题目：

1）、 查询身份证号码为440401430103082的申请日期

Select g\_applydate from g\_cardapply where applyno = (Select g\_applyno from

g\_cardapplydetail where g\_idcard = ‘440401430103082’);

或: Select a.g\_applydate from g\_cardapply a, g\_cardapplydetail b where

a.g\_applyno = b.g\_applyno and b.g\_idcard = ‘440401430103082’;

2）、查询同一个身份证号码有两条以上记录的身份证号码及记录个数

Select g\_idcard,count(\*) from g\_cardapplydetail

group by g\_idcard

having count(\*) > 1;

3）、 将身份证号码为440401430103082的记录在两个表中的申请状态改为07

Update g\_cardapplydetail set g\_state = ‘07’ where g\_idcard =

‘440401430103082’;

update a set g\_state = '07';

from g\_cardapply a,g\_cardapplydetail b where a.g\_applyno = b.g\_applyno

and b.g\_idcard = '440401430103082';

4）、 删除g\_cardapplydetail表中所有姓李的记录

Delete from g\_cardapplydetail where g\_name like ‘李%’;

1. 写出连接SQLSERVER数据库和ORACLE数据库的连接方法

orcale:

try

{

String name=""oracle.jdbc.driver.OracleDriver"";"

String url = ""jdbc:orcale:thin:@localhost:1521:kevin"";"

Class.forName(name).newInstance();

Conection con = DriverManager.getConnection(url,""kevin"",""kevin"");"

System.out.println(""连接 Oracle 9i成功！"");"

con.close();

}catch(Exception e)

{

e.printStackTrace():

}

SQL Server2000:

try

{

String name=""com.microsoft.jdbc.sqlserver.SQLServerDriver"";"

String url =

"jdbc:microsoft:sqlserver://localhost:1433:DatabaseName=JaveWeb";

Class.forName(name).newInstance();

Conection con = DriverManager.getConnection(url,""sa"",""sa"");"

System.out.println(""连接 SQL Server 2000成功！"");"

con.close();

}catch(Exception e)

{

e.printStackTrace():

}

1. 两个int型的变量a，b，不通过第三个变量，值互换，如何现实？

答:有两种解法, 一种用算术算法, 一种用^(异或)

a = a + b;

b = a - b;

a = a - b;

or

a = a^b;// 只能对int,char..

b = a^b;

a = a^b;

or

a ^= b ^= a;

1. 一个粗细均匀的长直管子，两端开口，里面有4个白球和4个黑球，球的直径、 两

端开口的直径等于管子内的内径，现在白球和黑球的排列是wwwwbbbb，要求不取出任何一个球，使得排列变为bbwwwwbb。

答：将管子两端接在一起，形成环路，倾斜后可行。

1. 有两根不均匀分布的香，每根香烧完的时间是一个小时。你能用什么方法来去顶

一段15分钟的时间？

答：首先点燃甲香两端及乙香一端，甲香燃烧完为半小时，接着点燃乙香的另一端，则

截止甲香燃尽后乙香烧完的这一时间段为十五分钟！

1. GC 是什么？为什么要有GC？

答：GC是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。Java程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc() ，但JVM可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在Java诞生初期，垃圾回收是Java最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今Java的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得iOS的系统比Android系统有更好的用户体验，其中一个深层次的原因就在于Android系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的Java进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java平台对堆内存回收和再利用的基本算法被称为标记和清除，但是Java对其进行了改进，采用“分代式垃圾收集”。这种方法会跟Java对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。

幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。

终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。

与垃圾回收相关的JVM参数：

#NAME?

#NAME?

-XX:-DisableExplicitGC --- 让System.gc()不产生任何作用

-XX:+PrintGCDetail --- 打印GC的细节

-XX:+PrintGCDateStamps --- 打印GC操作的时间戳

1. C++或Java中的异常处理机制的简单原理和应用。

答：1. 异常机制

1.1

异常机制是指当程序出现错误后，程序如何处理。具体来说，异常机制提供了程序退出的安全通道。当出现错误后，程序执行的流程发生改变，程序的控制权转移到异常处理器。

1.2

传统的处理异常的办法是，函数返回一个特殊的结果来表示出现异常（通常这个特殊结果是大家约定俗称的），调用该函数的程序负责检查并分析函数返回的结果。这样做有如下的弊端：例如函数返回-1代表出现异常，但是如果函数确实要返回-1这个正确的值时就会出现混淆；可读性降低，将程序代码与处理异常的代码混爹在一起；由调用函数的程序来分析错误，这就要求客户程序员对库函数有很深的了解。

1.3 异常处理的流程

1.3.1 遇到错误，方法立即结束，并不返回一个值；同时，抛出一个异常对象

1.3.2 调用该方法的程序也不会继续执行下去，而是搜索一个可以处理该异常的异常处理器，并执行其中的代码

2.异常的分类

2.1 异常的分类

2.1.1

异常的继承结构：基类为Throwable，Error和Exception继承Throwable，RuntimeException和IOException等继承Exception，具体的RuntimeException继承RuntimeException。

2.1.2

Error和RuntimeException及其子类成为未检查异常（unchecked），其它异常成为已检查异常（checked）。

2.2 每个类型的异常的特点

2.2.1 Error体系

Error类体系描述了Java运行系统中的内部错误以及资源耗尽的情形。应用程序不应该抛出这种类型的对象（一般是由虚拟机抛出）。如果出现这种错误，除了尽力使程序安全退出外，在其他方面是无能为力的。所以，在进行程序设计时，应该更关注Exception体系。

2.2.2 Exception体系

Exception体系包括RuntimeException体系和其他非RuntimeException的体系

2.2.2.1 RuntimeException

RuntimeException体系包括错误的类型转换、数组越界访问和试图访问空指针等等。处理RuntimeException的原则是：如果出现RuntimeException，那么一定是程序员的错误。例如，可以通过检查数组下标和数组边界来避免数组越界访问异常。

2.2.2.2 其他（IOException等等）

这类异常一般是外部错误，例如试图从文件尾后读取数据等，这并不是程序本身的错误，而是在应用环境中出现的外部错误。

2.3 与C++异常分类的不同

2.3.1

其实，Java中RuntimeException这个类名起的并不恰当，因为任何异常都是运行时出现的。（在编译时出现的错误并不是异常，换句话说，异常就是为了解决程序运行时出现的的错误）。

2.3.2

C++中logic\_error与Java中的RuntimeException是等价的，而runtime\_error与Java中非RuntimeException类型的异常是等价的。

3.异常的使用方法

3.1 声明方法抛出异常

3.1.1 语法：throws（略）

3.1.2 为什么要声明方法抛出异常？

方法是否抛出异常与方法返回值的类型一样重要。假设方法抛出异常确没有声明该方法将抛出异常，那么客户程序员可以调用这个方法而且不用编写处理异常的代码。那么，一旦出现异常，那么这个异常就没有合适的异常控制器来解决。

3.1.3 为什么抛出的异常一定是已检查异常？

RuntimeException与Error可以在任何代码中产生，它们不需要由程序员显示的抛出，一旦出现错误，那么相应的异常会被自动抛出。而已检查异常是由程序员抛出的，这分为两种情况：客户程序员调用会抛出异常的库函数（库函数的异常由库程序员抛出）；客户程序员自己使用throw语句抛出异常。遇到Error，程序员一般是无能为力的；遇到RuntimeException，那么一定是程序存在逻辑错误，要对程序进行修改（相当于调试的一种方法）；只有已检查异常才是程序员所关心的，程序应该且仅应该抛出或处理已检查异常。

3.1.4

注意：覆盖父类某方法的子类方法不能抛出比父类方法更多的异常，所以，有时设计父类的方法时会声明抛出异常，但实际的实现方法的代码却并不抛出异常，这样做的目的就是为了方便子类方法覆盖父类方法时可以抛出异常。

3.2 如何抛出异常

3.2.1 语法：throw（略）

3.2.2 抛出什么异常？

对于一个异常对象，真正有用的信息时异常的对象类型，而异常对象本身毫无意义。比如一个异常对象的类型是ClassCastException，那么这个类名就是唯一有用的信息。所以，在选择抛出什么异常时，最关键的就是选择异常的类名能够明确说明异常情况的类。

3.2.3

异常对象通常有两种构造函数：一种是无参数的构造函数；另一种是带一个字符串的构造函数，这个字符串将作为这个异常对象除了类型名以外的额外说明。

3.2.4

创建自己的异常：当Java内置的异常都不能明确的说明异常情况的时候，需要创建自己的异常。需要注意的是，唯一有用的就是类型名这个信息，所以不要在异常类的设计上花费精力。

3.3 捕获异常

如果一个异常没有被处理，那么，对于一个非图形界面的程序而言，该程序会被中止并输出异常信息；对于一个图形界面程序，也会输出异常的信息，但是程序并不中止，而是返回用Ы缑娲硌分小?BR> 3.3.1 语法：try、catch和finally（略）

控制器模块必须紧接在try块后面。若掷出一个异常，异常控制机制会搜寻参数与异常类型相符的第一个控制器随后它会进入那个catch

从句，并认为异常已得到控制。一旦catch 从句结束对控制器的搜索也会停止。

3.3.1.1 捕获多个异常（注意语法与捕获的顺序）（略）

3.3.1.2 finally的用法与异常处理流程（略）

3.3.2 异常处理做什么？

对于Java来说，由于有了垃圾收集，所以异常处理并不需要回收内存。但是依然有一些资源需要程序员来收集，比如文件、网络连接和图片等资源。

3.3.3 应该声明方法抛出异常还是在方法中捕获异常？

原则：捕捉并处理哪些知道如何处理的异常，而传递哪些不知道如何处理的异常

3.3.4 再次抛出异常

3.3.4.1 为什么要再次抛出异常？

在本级中，只能处理一部分内容，有些处理需要在更高一级的环境中完成，所以应该再次抛出异常。这样可以使每级的异常处理器处理它能够处理的异常。

3.3.4.2 异常处理流程

对应与同一try块的catch块将被忽略，抛出的异常将进入更高的一级。

4.关于异常的其他问题

4.1 过度使用异常

首先，使用异常很方便，所以程序员一般不再愿意编写处理错误的代码，而仅仅是简简单单的抛出一个异常。这样做是不对的，对于完全已知的错误，应该编写处理这种错误的代码，增加程序的鲁棒性。另外，异常机制的效率很差。

4.2 将异常与普通错误区分开

对于普通的完全一致的错误，应该编写处理这种错误的代码，增加程序的鲁棒性。只有外部的不能确定和预知的运行时错误才需要使用异常。

4.3 异常对象中包含的信息

一般情况下，异常对象唯一有用的信息就是类型信息。但使用异常带字符串的构造函数

时，这个字符串还可以作为额外的信息。调用异常对象的getMessage()、toString()或者

printStackTrace()方法可以分别得到异常对象的额外信息、类名和调用堆栈的信息。并且

后一种包含的信息是前一种的超集。

以下是还未学习知识点：

1. sleep() 和 wait() 有什么区别？

答：sleep()方法是使线程停止一段时间的方法。在sleep 时间间隔期满后，线程不一定立即恢复执行。这是因为在那个时刻，其它线程可能正在运行而且没有被调度为放弃执行，除非(a)“醒来”的线程具有更高的优先级。(b)正在运行的线程因为其它原因而阻塞。wait()是线程交互时，如果线程对一个同步对象x 发出一个wait()调用，该线程会暂停执行，被调对象进入等待状态，直到被唤醒或等待时间到。

1. 说出ArrayList,Vector, LinkedList的存储性能和特性

答：ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector由于使用了synchronized方法（线程安全），通常性能上较ArrayList差，而LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

1. jsp有哪些内置对象？作用分别是什么？

答：JSP共有以下9种基本内置组件（可与ASP的6种内部组件相对应）：

request 用户端请求，此请求会包含来自GET/POST请求的参数

response 网页传回用户端的回应

pageContext 网页的属性是在这里管理

session 与请求有关的会话期

application servlet 正在执行的内容

out 用来传送回应的输出

config servlet的构架部件

page JSP网页本身

exception 针对错误网页，未捕捉的例外

1. wait、notify不在synchronized方法或者synchronized block中被调用的后果是可以

编译，完全可以独立使用？

答：错误 可以编译，但运行时可能出错

1. 什么时候用assert？

答：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。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。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为true；如果表达式计算为false，那么系统会报告一个AssertionError。

断言用于调试目的：

assert(a > 0); // throws an AssertionError if a <= 0

断言可以有两种形式：

assert Expression1;

assert Expression1 : Expression2 ;

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用source 1.4 标记：

javac -source 1.4 Test.java

要在运行时启用断言，可使用-enableassertions 或者-ea 标记。

要在运行时选择禁用断言，可使用-da 或者-disableassertions 标记。

要在系统类中启用断言，可使用-esa 或者-dsa 标记。还可以在包的基础上启用或者禁用断言。可以在预计正常情况下不会到达的任何位置上放置断言。断言可以用于验证传递给私有方法的参数。不过，断言不应该用于验证传递给公有方法的参数，因为不管是否启用了断言，公有方法都必须检查其参数。不过，既可以在公有方法中，也可以在非公有方法中利用断言测试后置条件。另外，断言不应该以任何方式改变程序的状态。

1. 线程

答：能用java.lang,Thread和java.lang.Runnable两种方法定义，实例化和开始一个新的线程。

知道哪些情况下可能阻止一个线程的执行。

能使用synchronized,wait,notify和notifyAll去解决避免同时访问及其线程间相互通讯的问题。

当执行synchronized,wait,notify和notifyAll时，知道线程和对象锁之间的交互作用。

1. Static Nested Class 和 Inner Class的不同，说得越多越好(面试题有的很笼统)。

答：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。具体可见http: //www.frontfree.net/articles/services/view.asp?id=704&page=1

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

Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现

HashMap允许将null作为一个entry的key或者value，而Hashtable不允许

还有就是，HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。

最大的不同是，Hashtable的方法是Synchronize的，而HashMap不是，在

多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap

就必须为之提供外同步。

1. 启动一个线程是用run()还是start()?

答：启动一个线程是调用start()方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由JVM调度并执行。这并不意味着线程就会立即运行。run()方法可以产生必须退出的标志来停止一个线程。

1. 当一个线程进入一个对象的一个synchronized方法后，其它线程是否可进入此对

象的其它方法?

答：不能，一个对象的一个synchronized方法只能由一个线程访问。

1. EJB与Java Bean的区别？

答：Java Bean 是可复用的组件，对Java Bean并没有严格的规范，理论上讲，任何一个

Java类都可以是一个Bean。但通常情况下，由于Java Bean是被容器所创建(如Tomcat)

的，所以Java Bean应具有一个无参的构造器，另外，通常Java Bean还要实现Serializable

接口用于实现Bean的持久性。Java Bean实际上相当于微软COM模型中的本地进程内

COM组件，它是不能被跨进程访问的。Enterprise Java Bean 相当于DCOM，即分布式

组件。它是基于Java的远程方法调用(RMI)技术的，所以EJB可以被远程访问(跨进程、

跨计算机)。但EJB必须被布署在诸如Webspere、WebLogic这样的容器中，EJB客户从

不直接访问真正的EJB组件，而是通过其容器访问。EJB容器是EJB组件的代理，EJB组

件由容器所创建和管理。客户通过容器来访问真正的EJB组件。

1. 描述Structs体系结构？对应各个部分的开发工作主要包括哪些？

答：Struts 是MVC的一种实现，它将 Servlet和 JSP 标记（属于 J2EE 规范）用作实现的一部分。Struts继承了MVC的各项特性，并根据J2EE的特点，做了相应的变化与扩展。

Struts的体系结构包括模型（Model），视图（View）和控制器（Controller）三部分。

从MVC角度来看看struts的体系结构（Model 2）与工作原理：

1）模型（Model）

在Struts的体系结构中，模型分为两个部分：系统的内部状态和可以改变状态的操作（事务逻辑）。内部状态通常由一组ActinForm Bean表示。根据设计或应用程序复杂度的不同，这些Bean可以是自包含的并具有持续的状态，或只在需要时才获得数据（从某个数据库）。大型应用程序通常在方法内部封装事务逻辑（操作），这些方法可以被拥有状态信息的bean调用。比如购物车bean，它拥有用户购买商品的信息，可能还有checkOut()方法用来检查用户的信用卡，并向仓库发定货信息。 小型程序中，操作可能会被内嵌在Action类，它是struts框架中控制器角色的一部分。当逻辑简单时这个方法很适合。 建议用户将事务逻辑（要做什么）与Action类所扮演的角色（决定做什么）分开。

2）视图（View）

视图主要由JSP建立，struts包含扩展自定义标签库（TagLib），可以简化创建完全国际化用户界面的过程。目前的标签库包括：Bean Tags、HTML tags、Logic Tags、Nested Tags 以及Template Tags等。

3）控制器（Controller）

在struts中，基本的控制器组件是ActionServlet类中的实例servelt，实际使用的servlet在配置文件中由一组映射（由ActionMapping类进行描述）进行定义。对于业务逻辑的操作则主要由Action、ActionMapping、ActionForward这几个组件协调完成的，其中Action扮演了真正的业务逻辑的实现者，ActionMapping与ActionForward则指定了不同业务逻辑或流程的运行方向。struts-config.xml 文件配置控制器。

1. 简述StringTokenizer作用？

答：按照指定的分隔符，将字符串解析成若干语言符号。

1. 实现多线程的两种方法？

答： a) 从Java.lang.Thread类派生一个新的线程类，重写它的run()方法。 b) 实现Runnable接口，重写Runable接口中的run()方法。

1. 简述一个线程的生命周期？

答：新建---运行---中断---死亡

1. 如何让一个准备就绪的线程运行？

答：调用线程的start方法让一个处于准备就绪的状态运行。

1. 如何让一个线程休眠1000毫秒？

答：调用线程的sleep方法，参数为1000。

1. 如何使线程同步？

答： a)使用同步方法 synchronized void methodA() { } b) 使用同步块

synchronized(object) { //要同步的语句 }