90、八种基本数据类型（概率20%）

byte、short、int、long、float、double、char、boolean

31·&和&&的区别（概率10%）

一个位运算符，一个是逻辑运算符

具体代码中&&具有短路的作用(前面如果条件不成立,后面就不走了), &没有,

16·面向对象的特征？（概率60%）

**封装**

封装最好理解了。封装是面向对象的特征之一，是对象和类概念的主要特性。

封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

**继承**

面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

**多态**

多态性（polymorphisn）是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

实现多态，有二种方式，覆盖，重载。

**覆盖**，是指子类重新定义父类的虚函数的做法。

1:方法名相同,参数类型相同,子类的返回值类型大于等于父类,修饰符子类的访问权限要大于等于父类,子类抛出的异常不能大于父类.

**重载**，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。

1:同一方法中方法名相同,参数列表不同,参数的个数,类型,顺序(具体使用自定义adapter)

49.静态变量合实例变量的区别（概率30%）

语法：静态变量前要加static来修饰，而实例变量前则不加；

程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量；静态变量不属于某个实例对象，而是属于类，所以也称为类变量，静态变量就会被分配空间，静态变量就可以被使用了。

55.接口和抽象类的区别（概率40%）

abstract可以修饰抽象方法，而一个类只要有一个抽象方法，就必须用

abstract定义该类，即抽象类。

用interface修饰的类，里面的方法都是抽象方法，因此在定义接口的时候，可以

直接不加那些修饰，系统会默认的添上去。接口里面的字段都是公有常量，即

public static final修饰的字段。

//接口是多继承 、抽象类是单继承

//接口里面只有抽象方法

// 抽象类: 有抽象方法和普通方法 、 抽象类==父类( 抽象类是 顶级父类 )

// 比较两个对象是否相等 == 比较的是地址、 = 比较的是值

//final不能和 static 、actaret 一起使用

51.什么是匿名内部类，在什么时候调用（概率30%）

内部类：内部类可以是static的或者非static的，static内部类只能包含静态方法和静态类变量，只能访问外部类的静态元素，内部类可以实例化，多次使用。

匿名内部类：它只能使用一次，不区分static和非static。如果用到外部类的变量的话，必须是类变量或者实例变量，就是必须是类定义的变量，或者final的局部变量。匿名内部类如果是继承某个类的话是可以重写那个类的方法的，这个和普通内部类一样。

实现事件监听器的时候

用匿名内部类编码非常简洁，也容易读懂，不重复利用时使用。

94 public，private，protect权限（概率40%）

public在任何情况下都可用；

protect可在当前类，同一个包中和子孙类中使用；

default：可在当前类和同一个包中使用

private只能在当前类中使用

19·Final，finally，finalized，区别；（概率30%）

final用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。(geitview内部类访问外部类的时候需要加final)

finally是异常处理语句结构的一部分，表示总是执行。(捕获异常的时候用过)

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此

方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。

61.Throws与Throw的区别（概率20%）

1.throw则是指抛出的一个具体的异常类型。

2.通常在一个方法（类）的声明处通过throws声明方法（类）可能抛出的异常信息，而在方法（类）内部通过throw声明一个具体的异常信息。

3.throws通常不用显示的捕获异常，可由系统自动将所有捕获的异常信息抛给上级方法；

throw则需要用户自己捕获相关的异常，而后在对其进行相关包装，最后在将包装后的异常信息抛出。

如果在方法中会有异常被抛出而你又不希望在这个方法体内对此异常进行处理，可以使用throws在声明方法的时候同时声明他可能会跑出的异常。

1. 两者位置不同.

92、异常（概率40%）

答：编译时异常：程序正确，但因为外在的环境条件不满足引发。例如：用户错

误及I/O问题----程序试图打开一个并不存在的远程Socket端口。这不是程序本

身的逻辑错误，而很可能是远程机器名字错误(用户拼写错误)。对商用软件系统，

程序开发者必须考虑并处理这个问题。Java编译器强制要求处理这类异常，如果

不捕获这类异常，程序将不能被编译。

运行期异常：这意味着程序存在bug，如数组越界，0被除，入参不满足规范.....

这类异常需要更改程序来避免，Java编译器强制要求处理这类异常。

93、==equals区别（概率40%）

1、 ==是判断两个变量或实例是不是指向同一个内存空间

equals是判断两个变量或实例所指向的内存空间的值是不是相同

2、==是指对内存地址进行比较

equals()是对字符串的内容进行比较

3、==指引用是否相同

equals（）指的是值是否相同

79、集合的实现类（概率40%）

HashSet：元素无需、不可重复；ArrayList：元素有序，可重复

HashMap以键值对的形式存取数据，key值不可重复，value值可重复

8.ArrayList和LinkedList区别？（概率30%）

存数据，ArrayList**数组存储数据**，索引值以下标来搜索，**查询比较方便，删除增加比较麻烦**，

但是linkedList以**链表式存储数据**，对于增删比较方便。

可以这样说：当操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能；

当你的操作是在一列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用LinkedList了。

ArrayList和LinkedList在性能上各 有优缺点，都有各自所适用的地方，总的说来可以描述如下：

对ArrayList和LinkedList而言，在列表末尾增加一个元素所花的开销都是固定的。对 ArrayList而言，主要是在内部数组中增加一项，指向所添加的元素，偶尔可能会导致对数组重新进行分配；而对LinkedList而言，这个开销是 统一的，分配一个内部Entry对象。

在ArrayList的 中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动；而在LinkedList的中间插入或删除一个元素的开销是固定的。

LinkedList不支持高效的随机元素访问。

ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间

78、String,StringBuffer区别（概率30%）

JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，

即包含多个字符的字符数据。这个String类提供了数值不可改变的字符串。而这个

StringBuffer类提供的字符串进行修改。

StringBuilder与StringBuffer的区别 。

StringBuffer

1，一个类似于 String 的字符串缓冲区，对它的修改不会像String那样重创建对象。

2，使用append()方法修改Stringbuffer的值，使用toString()方法转换为字符串。

Stringbuild是jdk1.5后用来替换stringBuffer的一个类，大多数时候可以替换

StringBuffer。和StringBuffer的区别在于Stringbuild是一个单线程使用的类，不值

执行线程同步所以比StringBuffer的速度快，效率高。

82.io流读取输出方法（概率20%）

文件读取：

public static void main(String[] args) throws IOException

{

//创建一个文件读取流对象，并初始化时指定名称的文件相关联。

//要保证此文件是已存在，如不存在会发生异常：FileNotFoundException

FileReader fr = new FileReader("demo.txt");

//调用读取流对象的read方法。

//read方法一次只读一个字符，而且会自动向下读取。当读取完后会返回-1.

int ch = 0;

while((ch = fr.read())!=-1)

System.out.print((char)ch);

fr.close();

}

53.什么是线程（概率60%）

线程，有时被称为[轻量级进程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=65949654" \t "http://blog.csdn.net/dangai0201/article/details/_blank)(Lightweight Process，LWP)，是程序执行流的最小[单元](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=275431" \t "http://blog.csdn.net/dangai0201/article/details/_blank)。一个标准的线程由线程[ID](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=265569" \t "http://blog.csdn.net/dangai0201/article/details/_blank)，当前指令指针(PC)，[寄存器](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=35071" \t "http://blog.csdn.net/dangai0201/article/details/_blank)集合和[堆栈](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=331232" \t "http://blog.csdn.net/dangai0201/article/details/_blank)组成。另外，线程是[进程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=288622" \t "http://blog.csdn.net/dangai0201/article/details/_blank)中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有[系统](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=349837" \t "http://blog.csdn.net/dangai0201/article/details/_blank)资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以[创建](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=100768" \t "http://blog.csdn.net/dangai0201/article/details/_blank)和[撤消](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=126328" \t "http://blog.csdn.net/dangai0201/article/details/_blank)另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有[就绪](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=425221" \t "http://blog.csdn.net/dangai0201/article/details/_blank)、[阻塞](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=646488" \t "http://blog.csdn.net/dangai0201/article/details/_blank)和[运行](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=493544" \t "http://blog.csdn.net/dangai0201/article/details/_blank)三种基本状态。每一个程序都至少有一个线程，那就是程序本身。

线程是程序中一个单一的顺序控制流程。在单个程序中同时运行多个线程完成不同的工作，称为[多线程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=452880" \t "http://blog.csdn.net/dangai0201/article/details/_blank)。

54.线程和进程的区别（概率60%）

// 线程与进程的区别:

定义:

1. 进程是系统进行资源分配和调度的一个独立单位
2. 线程是进程的一个实体，是CPU调度和分派的基本单位，线程自己基本上不拥有系统资源，只拥有一点在系统运行中必不可少的资源，但是它可与同属一个进程的其他线程共享进程所拥有的全部资源

关系:

1. 一个线程可以创建和撤销另一个线程，同一个进程中的多个线程可以同时并发执行，他可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行顺序

区别:

进程和线程区别: 他们是不同的操作系统资源管理方式。

进程有独立的地址空间，一个进程崩溃后，不会对其他进程产生影响

线程只是一个进程中不同的执行路径，线程有自己的堆栈和局部变量，线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉， 所以多线程的操作要比多进程的程序健壮。 但在进程切换时耗费资源较大，效率要差一些

对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程，（因为进程在切换时耗费的资源要大一些，效率差一些）

进程是表示资源分配的基本单位，又是调度运行的基本单位。

线程是进程中执行运算的最小单位，亦即执行处理机调度的基本单位。

线程的优点：

（1）易于调度。

（2）提高并发性。通过线程可方便有效地实现并发性。进程可创建多个线程来执行同一程序的不同部分。

（3）开销少。创建线程比创建进程要快，所需开销很少。。

（4）利于充分发挥多处理器的功能。通过创建多线程进程（即一个进程可具有两个或更多个线程），每个线程在一个处理器上运行，从而实现应用程序的并发性，使每个处理器都得到充分运行。

进程和线程的关系：

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。 （2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。

（3）处理机分给线程，即真正在处理机上运行的是线程。

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

47.Sleep和wait的区别（概率30%）

1.这两个方法来自不同的类分别是，sleep来自Thread类，和wait来自Object

类。

2.最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可

以使用同步控制块或者方法。sleep不出让系统资源；wait是进入线程等待池

等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，

因为如果wait线程的运行资源不够，再出来也没用，要等待其他线程调用

notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系

统资源。sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到

只能调用interrupt()强行打断。

3.wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而

sleep可以在任何地方使用

Sleep需要捕获异常,而wait不需要

72、说说java的反射机制原理以及应用场合（概率40%）

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有

属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态

获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

类对象: 封装了类的描述信息的对象,类加载的产物,由jvm创建java.lang.Class

应用场景: Gson 序列化反序列化

获取类对象的方式:1. 类名.class2. 类的对象.getClass()3. Class.forName("

包名.类名")

74、java有几种引用类型（概率40%）

强引用，软引用、弱引用、幽灵引用、

68.gc是什么（概率70%）

GC是垃圾收集的意思（Gabage Collection）,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。

97、什么时候用递归（概率30%）

树用递归方便，用其它方法好像跟本不可能．

递归用在特定的函数如 f(x)= f(x-1) + 2

像这种情况，你想算出f(x)就必需算出f(x-1)，而f(x)和f(x-1)实际上都是用共一个方法，只是参数相差一．这种时候用递归就很快．

98、jdbc中怎么处理事物（概率50%）

事务，也是数据库事务，指的是作为单个逻辑工作单元执行的一系列操作。正常的情况下，操作应该顺利进行，与操作相关的所有数据库信息也成功地更新；

但是，如果在这一系列过程中任何一个环节出了差错，导致操作失败了，数据库中所有信息都必须保持操作前的状态不变。否则，数据库的信息将会一片混乱而不可预测。

一个逻辑工作单元要称为事务，必须满足ACID（原子性，一致性，隔离性和持久性）

事务的结束只能有两种形式：提交和回滚。操作完全成功则提交，产生永久性的修改；操作不完全成功则回滚，恢复到事务开始前的状态。它们将结束一个事务。

（1）关闭自动提交事务。通过设置连接的自动提交事务属性为false，