**Java对象与面向对象**

java中“万物皆对象”，使用new关键字来在堆中创建一个引用对象。Java中使用class来修饰一个类，如人类（抽象的概念），而每一个具体的人就是一个个的对象。

java中数据的存储位置：1.寄存器（register）最快的存储区，位于处理器的内部，不能被直接控制。

2.堆栈（stack），位于RAM中，通过“堆栈指针”来操作，向下移动分配内存，向上移动释放内存。用于存放基本数据（int，。。），对象的引用数据。内存数据共享。读写速度块，只比寄存器慢。

3.堆（heap），一种通用的内存池，也位于RAM中，来存放java对象。内存空间分配较为灵活（new），释放有java垃圾回收器来回收。

4.静态存储（static storage），存储在固定的位置，也在RAM中，必须是static修饰的。

5.常量存储区域（constant storage），位于程序代码内部，不能被更改，有的存储在ROM中。

6.非RAM存储，位于程序代码之外的，如：流对象，持久化对象（实现了serializable接口），用于网络传输，存储与本地。

3) 常量池：位于方法区，用于保存在编译期间已确定的数据，包括关于类，方法，接口等的常量，也包含字符串常量，static final修饰的常量，一些包装类（不包含浮点型）范围在-128~+127之间的数据。

4) String a=new String(“b”);创建了几个对象？创建两个对象a,b。首先在java栈中分配空间创建a引用，然后在java堆中创建b引用，接着在常量池中查找有没有该常量“b”，如果有的话直接指向常量池中的b,如果没有则在常量池中去创建该常量“b”

5)栈内存共享：int a=1,b=1;,过程：首先在java栈中创建a,b的引用，然后在栈中寻找“1”数据，如果有的话，a,b指向它，如果没有则去创建。

6) 包装类Integer a=1; Integer b=1; a==b,true; 常量池

Integer a=128; Integer b=128; a==b,fasle; 范围在-128~+127之外

String a=”1”; String b=”1”; a==b,true; 常量池

String a=new String(“1”); String b=new String(“1”); a==b,false; 不同的对象；

7) JVM的结构：直接内存（NIO），程序计数器，堆，栈，本地方法栈（native服务），方法区（常量池）。

**Java操作符与流程控制**

1. 算数操作符+，-，\*，/，%五种运算符。当然还有简写的运算符如+=，-=，\*=，/=，%=。
2. 一元加，减操作符，表示正负号。递增递减运算符++，--。
3. 关系操作符，结果是一个布尔类型结果。>,<，<=,>=,==,!=。
4. 逻辑操作符，结果也是一个布尔型结果。&&，||，!，（双）与或非。
5. 位操作符，进行bit（比特）运算，&，|，!，^（异或）
6. 移位操作符，操作的也是二进制的“位”，只能用来处理整数类型（byte，short，int，long）。左移位操作符（<<）左边的数向左移动右侧指定的数值。低位补0；

右移位操作符（>>）左边的数向左移动右侧指定的数值。高位正数补0，负数补1；

右移位操作符（>>>）无论正负，高位补0；

1. 三元操作符，条件表达式？值1：值2；
2. 逗号操作符，用于for循环里面
3. 流程控制，for，while，dowhile,switch。其中for和while一样知道条件为假停止，而do-while至少执行一次，switch位开发判断（jdk 5.0支持string类型）
4. 关键字return，如果有方法中有返回值则返回数值并结束方法，没有返回值则直接结束方法。
5. 关键字break，结束当前循环
6. 关键字continue，结束本次循环，开始下次循环。
7. 关于java中的“goto”用法，break 标签；和continue 标签；标签写在循环的上面类似“outer：”，（尽量少用，除非有多个嵌套）。

**Java的初始化与清除**

1. 方法重载，只可以根据参数列表。在同一个类中，方法名相同，参数列表的个数，顺序，参数类型不同。（tips:不能根据返回值来判断方法重载）
2. this关键字表示当前对象，本类对象
3. static静态的，只可以通过类来调用，静态的方法不可以调用静态的方法，反之可以。
4. java中资源的释放清除有垃圾回收器（garbage collection）来进行处理，其中object中的finalize()方法，只是建议gc去回收资源，有可能不被回收。
5. 垃圾回收机制：引用计数器（简单速度慢的技术），在每个引用都有一个计数器，new了之后计数器加1，当引用被置为null时，计数器减1，垃圾回收器遍历后释放资源。
6. 成员变量可以默认初始化，局部变量不能自动初始化。
7. 如果没有给成员变量赋值，而是在构造中初始化。则先是系统默认初始化，然后才在构造器中再次赋值。
8. 类初始化的顺序，静态变量>静态代码块>普通变量>普通代码块>构造器。

**Java中隐藏具体实现**

1. package关键字位于位于程序的首行，用于指定一个程序的包名（命名字母小写），这个样才能使所有人编写的java类不会有一致的重复的。
2. import关键字用来将程序库中的某个类或者所有类进行导包，也可以在本类中使用完整包名+类名的方式。
3. 如果程序中不同的包名里有相同的类名，导包时产生冲突，则直接使用完整包名+类名的方式。
4. 四种访问权限由大到小：public(所有的都可以访问)>protected(继承，包访问)>default(无关键字，默认不写，包访问权限)>private(类访问权限)。
5. 类访问权限只能是包访问权限和public的，不能是private（这样会使得除该类之外再无其他类可以访问它，不能被其他类使用）和protected的。如果不想让其他类来访问该类，只需让该类的构造器为private，禁止访问即可。

**Java中的复用类**

1. 组合语法：1.在定义对象的地方进行初始化。2.在类的构造器中。3.在需要的时候进行初始化，这种方式称之为“惰性初始化”（lazy initialization）
2. Java中的继承，如果父类有无参数的构造器，在创建基类的对象时，会默认隐式调用父类的无参构造器。如果父类有参数构造器，并且没有无参数的构造器，则子类要显式的调用父类的有参构造器，使用super(…)（且要放在第一位）。并且调用的顺序是先调用父类的构造器，在调用子类的构造器。
3. final关键字表示不可修改的，不可变的，可以用来修饰类，方法，变量。
4. final关键字修饰类，表示此类不可被继承。修饰方法，表示此方法不可被重写override。修饰变量，表示此变量不可被更改。
5. static final和final并无重大的区别，static final表示整个类公用一处内存。
6. final修饰的成员变量，系统不会默认赋值，需要我们自己赋值。是编译期的修饰符，程序在编译时期就可以知道它的值。（并不一定对，final的赋值为new Random().nextInt()一个随机值。）
7. blank final即空白final，空白final修饰的成员变量可以不赋值，但必须在构造器中初始化，即final数据必须要在使用前被赋值。
8. final修饰参数，表示参数的引用固定，并且不能被修改其值，只允许读取；
9. final修饰方法，一表示方法不能被重写，二也提高了效率（在编译器指明此方法是final）。
10. final和private修饰方法时，其实类似，private的方法其子类不能重写，有隐式final的意思。
11. final修饰类，意味着此类不能被继承，其中的方法也是隐含final的，因为你不能重写他。

**Java中的多态**

1. 绑定：讲一个方法调用同一个方法主题关联起来被称作绑定；
2. 前期绑定（静态绑定）：若程序执行前，进行绑定（编译时期），static 和final
3. 后期绑定（动态绑定）：在运行时，根据对象的类型进行绑定。除了static和final方法，都是后期绑定。（final方法，效率提高，关闭动态绑定）
4. 方法重写：多态只会调用子类中重写的方法，不会调用子类特有的方法。Final和private的方法都不能被重写，private隐含final含义，继承不能继承private属性。

**Java中的接口与内部类**

1. 接口：interface是一个“纯粹的”抽象类，允许创建只有方法名，返回值，方法名，无方法体的方法（默认是public），和数据成员（默认是public static final的）。接口是is-like-a 的情形。
2. 接口之间可以多继承来进行扩展功能。
3. 内部类：嵌套在class中的其他类，分为四种，成员内部类，方法内部类，静态内部类（嵌套类），匿名内部类。
4. 内部类标识符，每一个类编译后都会生成一个.class文件，而有内部类的类，编译后生成，类名$内部类名.class.
5. 为什么要使用内部类？内部类解决了“多重继承”的问题，接口解决了部分问题，而每个内部类都能独立的继承自一个类。
6. 接口修饰符只能是public和abstract

**Java中的异常与错误处理**

1. 异常的根类为Throwable，其子类位Error和Exception.
2. 异常的形式参数：1.无参构造 throw new NullPointerException();

2.有参构造 throw new NullPointerException(“t==null”);

1. 异常的捕获与抛出：1.捕获，使用try..catch(Exception e)..catch..finally

2.抛出，让它的调用者来进行处理，是在方法名后面使用throws关键字，throws Exception..,。

1. 异常的方法：1.getMessage(),相当于类中的toString()方法，打印构造msg信息

2.printStackTrace(),追踪栈的轨迹，打印详细信息

3.toString(),返回对Throwable的简单描述，包含异常信息。

4,.fillInStackTrace(),用于在Throwable对象内部记录栈框架的当前状态。

1. 异常的重新抛出，在catch块里面使用关键字throw e,其后续的catch语句将不会被调用。
2. 异常链：在捕获一个异常的时候，抛出另一个异常，并且把原始异常的信息保存下来。Java中Throwable的三个子类Error,Exception,RuntimeException提供了带有cause的构造器，通过将原有的异常传递给新的异常。而其他异常我们可以使用initCause()方法而不是构造器来传递异常。
3. 运行时异常（RuntimeException），是在运行期间JVM帮我们抛出，最终直达main方法，在程序退出时，调用printStackTrace()方法。所以不需要我们进行捕获处理。
4. finally表示程序一定会执行的程序块，用于在最后处理释放资源，关闭资源。
5. 异常丢失：当使用try…finally..的时候，并且在try和finally中分别抛出异常，只会有finally的异常被打印。
6. 异常的匹配：1.java中的异常（catch块）匹配采用“就近原则“，如果最近的匹配，就会处理掉，不会执行后面的catch语句。

2.java中的异常匹配并不是需要完全一致，匹配其父类也可以。

**Java中的类型检查**

1. 运行期类型识别（RTTI，run-time type identification），必须是在编译时期，就知道所有的类型，编译器在编译期打开和检查.class文件。
2. 类加载的顺序，是先编译后生成.class文件，然后JVM寻找同名的.class文件，加载到内存中，然后，才可以使用。
3. 获取Class的三种方法，1.Class.forName(“java.lang.Integer”);

2.Integer.class;（推荐使用，更加高效，安全，编译期受到检查）

3.Integer.getClass();

1. 判断对象是否可以向下转型方式：1.instanceOf 对象 instanceOf 类

2.isInstance(Object o) <Class>.isInstance(对象)；

1. Java中的反射：编译器在编译期间并不检查和加载.class文件，而是在运行期打开和检查class文件。（通过Class获取method，constructor,etc.）
2. 如何判断char数组是基本数据类型还是对象？通过反射的方式，排除法判断，isPrimitive()是否是基本数据类型，isEnum()是否是枚举类型，isArray()是否是数组类型。。所以他是引用数据类型{类，接口，数组}，是对象。

**Java中的数组**

1. 数组：是一种效率最高的存储和随机访问对象引用序列的方式。数组的数据结构是线性结构序列，元素访问非常迅速，只有数组可以保存基本数据类型，而容器则不能。
2. 数组是一个引用类型，是一个对象。在栈中保存引用，在堆中开辟空间，保存数据。
3. java.util.Arrays类是数组的静态工厂类，提供了很多数组操作的方法，如toString()打印所有数组，equals比较数组是否相等，sort排序，fill插入，binarySearch查找，asList（）数组转list.
4. 数组复制：1.使用Arrays.copy(); 返回一个新的数组

2.使用System.arraycopy() 在原有数组的基础上，进行复制，为所有类型做了重载。

1. 数组排序：1.升序：Arrays.sort(),默认升序。

2.降序： Arrays.sort(Object[] o,Comparator c); 一种使用Collections.resverOrder()。另一种使用自己定义的比较器implement Comparator。

1. Comparable接口与comparator接口的区别。Comparable接口是排序接口，内部一个方法compareTo(),集合内部排序实现的接口，如自定义的类，在集合中排序。而Comparator接口则是外部排序，是一个比较器，使用策略的模式，进行排序，内部两个方法，一个equals(不需要实现，object已经实现)，一个compare()。
2. 容器：接口对象Collection,其子类List表示有序的可重复的，子类Set表示不可重复的，无序的。Map则是一种键值对的关系表。
3. Iterator:迭代器，主要使用集合调用iterator()方法返回Iterator对象，使用hasnext()判断是否还有对象，使用next()方法返回迭代的数据，使用remove()方法可以移除迭代对象。
4. toString()问题：将对象加入list，直接输出获得的对象，则会默认重载对象的toString()方法，如果要打印地址的话，不能再对象的重载toString()中使用this来打印，否则会重复调用toString()造成递归，栈溢出。应该使用super.toString().
5. ArrayList与LinkList区别：arraylist是一种动态数组结构，当数组空间不够时再次分配内存进行数组构建，然后复制原有的数组，查询快，增删较慢（因为要移动数据，连续的内存空间）。而LinkList是一种链表结构，增删快，查询慢（因为要移动指针，离散的内存空间存储地址），使用于栈，队列和双向队列。
6. linkList实现栈和队列。
7. Set必须保证元素唯一，要实现equals()方法。

子类：HashSet,为了快速查找设计的Set,存入的对象必须定义hashCode()

TreeSet,保持次序的Set,底层为树结构，排序的话，必须实现Comparble接口。

LinkHashSet,具有HashSet的查询速度，内部用链表维护元素的顺序。

1. Map：子类：HashMap,给予散列表实现（取代Hashtable）

LinkedHashMap,类似于HashMap,迭代遍历时，取得该键值对的顺序，是插入次序，比hashmap慢一点，但是迭代访问时更快，链表结构

TreeMap,基于红黑树数据结构实现，会排序，

WeakHashMap,map中的对象也可以被释放

IdentityHashMap,使用==代替equals()对“键值对”作比较的hashmap.

1. Map的键值问题，如果要使用自己定义的类作为map的“键”，那么这个对象必须重载hashCode()和equals()方法，因为默认的equals()只是比较地址而已。
2. Map的数据结构是散列表，也就是数组+链表结构。其中“键”对象通过hashcode(),生成散列码，将其最为数组的角标索引，但是也有可能不用的键值生成相同的散列码，而数组中存放的也不是“值”，而是值的LinkedList，然后通过equals()方法进行比较。
3. HashSet和HashMap必须重写hashCode()和equals()方法，其中hashCode()不必是惟一的，但是要和equals()组合是唯一的。

如何重载？

hashCode(){ result = 17(非零值)，result=37\*result+c.hashcode() +id（成员变量） }

equals(Obect o) {return (o instanceOf A && c.equals((A)o.c)&& id == (A).id)}

1. A
2. Collections类中的实用方法：

1,max(Collection c)和min(Collection c)返回集合汇总最大最小的值（通常是自然排序）

2.max(Collection c,Comparator c)和min(Collection c,Comparator c)，比较器比较

3. 对于“unmodifiable”类型的方法，设置集合为“只读”，如果要修改的话，会报异常UnsupportedOperationException。

1. 4. arrayList,LinkedList,HashSet，HashMap…都不是线程同步的，所以要在多线程中使用，则要加一层包装，使用 1.List Collections.synchronizedList(..), Set Collections.synchronizedSet(..)

3.Map Collections.synchronizedMap(..)

1. 快速报错：如果在已经取得迭代器的情况下，再次向容器中添加数据，则可能造成异常ConcurrentModificationException.

**Java中的I/O系统**

1. 文件类File的基本使用，如创建file对象，获取file对象以及一些file信息，

大小length，路径getPath()，读写canRead()，目录过滤FilenameFilter&FileFilter，

文件最后一次修改时间lastModified()

1. InputStream类型：1.ByteArrayInputStream=允许将内存的缓冲区当inputStream使用

2.StringBufferInputSream 将String转换成InputSream

3.FileInputSream 用于从文件中读取信息

4.PipedInputSream 产生用于写入相关的PieedOutputSream的数据，实现“管道化”概念，用于线程之间的通信

5.SequenceInputStream 将两个或者多个InputStream对象转换成单一的InputSream

6.FilterInputSream 抽象类，作为修饰器的接口，其中，修饰器为其他的InputStream类提供功能

数据源：1.字节数组

2.String对象

3.文件

4.管道

5.一个由其他种类的流组成的序列

6.其他数据源

1. OutputSream类型：和以上输入对应
2. 标准输入输出流：System.out ,System.err返回PrintStream,打印流

但是System.in 返回的时InputSream,使用时需要进行包装。

1. 标准输入输出流的重定向：1.System.setIn(InputStream),重定向到控制台输出

2.System.setOut(PrintStream),重定向到另一个文件中

3.System.setErr(PrintStream),重定向到另一个文件中

1. NI/O：新的io流，在jdk1.4的时候进行了更新，其中旧的I/O库有三个类被改进了，用于获取FileChannel对象，FileInputStream,FileOutputStream,RandomAccessFile.
2. 压缩与解压缩：1.checkedInputSream,为任何的inputstream产生检验

2.CheckedOutputStream，为任何outputStream产生校验

3.DeflaterOutputStream,用于压缩的基类

4.ZipOutputStream,将数据压缩成zip

5.GZIPOutputStream 将数据压缩成GZIP格式

6.InflaterOutputStream,用于解压缩的基类

7.ZipInputStream,InflaterInputStream,解压缩Zip文件

8,GZIPInputStream,解压缩GZIP文件数据

1. 使用GZIPOutputStream和inputStream进行简单压缩与解压。
2. 使用ZIP进行多文件压缩，解压。

压缩： 1.先创建一个FileOutputStream f，创建一个压缩后的文件格式，文件名。

2. 先使用CheckedOutputStream(f,new Adler()) cos 为输出流产生校验

3．创建ZipOutputStream(cos),然后传入BufferedOutputStream中。

4.将要压缩的文件数组，进行遍历，然后读取，使用上面创建的输出流写入，生成zip文件。

解压缩：1.创建FileInputStream打开要解压的文件

2.创建checkedInputStream传入上述流，创建校验输入流

3.创建ZipInputStream zip，传入上述流，在创建BufferedInputStream传入上述流。

4.创建ZipEntry对象，使用zip.getNextEntry()循环不为空时，代表多个被压缩的对象。ZipEntry对象包含文件被压缩的时间，大小等等。

5.在上述循环中，在创建循环使用bufferedInputStream读取文件，输出。

解压缩单个文件：使用ZipFile对象来进行获取文件。

1. Jar文档：jdk自带的压缩工具。 Jar [options] destination [manifest] inputfile(s)
2. jar cf hh.jar \*.class 将所有的.class文件打包压缩为hh的jar文件
3. jar tmf hh.jar manifestFile.mf \*.class 创建清单文件
4. 对象序列化：将实现了Serializable接口的对象转换成一个字节序列，并在以后可以恢复称为原来的对象吗，可以再网络传输。
5. 对象序列化是面向字节的，使用ObjectInput/OutputStream,来对实现serialiiable接口的对象进行读写。
6. 序列化的控制：如果本类中有部分不想被序列化的，则可以通过实现Externalizable接口（是Serializable子类）来代替Serializable接口。默认提供两个方法，writeExternal(ObjectOutput out) 和readExternal(ObjectInput in)，一个是序列化写入，一个是反序列化读出，通过方法中的writeExternal()我们可以自己选择要序列化的部分和反序列化的部分。

注意：实现此方法时，会默认调用类的无参构造方法，不可不写，而且也要为public不能为default，否则报错。

1. 使用transient关键字来实现序列化控制，凡是此修饰的变量都不会被序列化。和Serializabel接口使用
2. Serializable接口tips:实现此接口的类，默认在序列化时会调用private void writeObject(ObjectOutputStream stream)，反序列化时会调用private void readObject(ObjectInputStream stream)，所以也可以使用这两个方法进行序列化控制。
3. 正则表达式：^表达式开头 \n换行符，

￥表达式结尾 \r回车

\*表示0个或多个 \t制表符

+表示1个或者多个 \f换页

？表示0个或者1个 \e escape (以上这些都不用转义)

.表示任何字符，\s 各种空格换行符，\S 非各种空格换行符 ，\d 数字 \D 非数字

\w 字符a-zA-Z0-9, \W非字符

1. 使用Pattern.complie(“”).matches(“”,””);或者Pattern.complie(“”).matcher().matches();返回boolean判断格式一致。
2. 使用find()表示向下迭代第一个匹配的字符，group()返回第一个匹配的字符.

而find(int i)表示指定角标匹配迭代。Group()组默认返回0组即，全部返回

**Java中的并发**

1. 线程的创建方式：1.继承java.lang.Thread类 2.实现Runnable接口
2. 想成让步：yield()方法，可以给线程调度（是抢占式的）一个暗示，你的工作做完了，可以让出处理器使用权了，但是这只是一个暗示，并不一定会接受。
3. 休眠：sleep(long l)使线程停止执行一段时间。.
4. 等待：join(long l)等待线程死亡。线程等待一段时间，其他线程运行在等待一段时间。

也可以在一个线程的基础上调用join()方法，此线程会挂起，等到目标线程死亡，才能恢复，或者调用interrupt()方法中断。

1. 使用setPriority(int priority)设置优先级。
2. 后台线程：daemon，通过设置setDaemon(true),所有非后台线程停止时，程序也就终止了，main就是个非后台线程。当程序中只有后台线程时（除了main()），jvm就会退出了。
3. 线程冲突：Jvm中一个线程有一个工作内存，线程共用进程中的内存并拷贝到自己的工作内存中，也就是说每个线程都有一个主内存中的变量副本，当每个线程的拷贝变量被修改时，要再次写入主内存中，这样就容易造成线程间的冲突。

使用关键字volatile，可以使得变量被更改时，刷新，更新所有持有该变量的线程（可见性），轻量的同步操作。

1. 同步控制：使用synchronized关键字，可以给方法加锁（锁是当前对象），也可以给类加锁（防止并发访问静态变量）
2. 原子操作：就是指不会被线程调度打断的操作，不需要加锁，一旦开始到结束，中间不会有任何的线程切换。
3. 线程的状态：1.新建（new）:只要new一个thread对象

2.就绪（runnable）：只要调度程序把时间片分配给线程，线程就可以运行了，也就是说线程可以运行也可以不被运行。

3.死亡（dead）：从run()方法中返回

4.阻塞（blocked）：线程处于阻塞时，调度机制将忽略线程，不会分配给他线程，除非他重新就绪。

1. Sleep()方法不会释放锁，而wait()方法可以释放锁。
2. 线程之间的通信：使用管道进行输入输出（pipe），使用PipedWriter和PipedReader进行管道通信，在管道中一个线程写入一个数据，另一个线程读取一个数据。如果线程读取时发现没有数据时，则将自动阻塞，相当于生产者消费者。
3. 线程间的死锁：就是当一个线程等待另一个线程时，而后者又在等待别的线程，最后有个线程在等在第一个线程，形成循环，死锁。

产生的条件：1.互斥条件：线程中至少一个资源不能被共享。

2.至少有一个线程持有一个资源，并等待获取另一个被占据的资源。

3.资源不能被线程抢占

4.必须有循环等待。

解决：只要以上一个问题即可，第4点最容易打破。

1. 线程正确的停止方法：使用flag退出run方法。不能使用stop()等废弃的方法，应为其不释放锁，易造成死锁。