# ****1、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。提供继承的类叫父类（超类、基类）、得到继承的类叫子类（派生类）。

- 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。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。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）

# ****2、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 | √ | √ | √ | √ |
| protected | √ | √ | √ | × |
| default | √ | √ | × | × |
| private | √ | × | × | × |

# ****3、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）。

# ****4、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;。

# ****5、short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗？****

答：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。

# ****6、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了基本数据类型，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Java 5开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

**class** AutoUnboxingTest {

**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

    }

}

最近还遇到一个面试题，也是和自动装箱和拆箱有点关系的，代码如下所示：

**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对象引用，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

**public** **static** Integer valueOf(**int** i) {

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

**return** IntegerCache.cache[i + (-IntegerCache.low)];

**return** **new** Integer(i);

}

简单的说，如果整型字面量的值在-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。

# ****7、&和&&的区别？****

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。

# ****8、解释内存中的栈(stack)、堆(heap)和静态区(static area)的用法。****

栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间

String str = **new** String("hello");

上面的语句中变量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区。

# ****9、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。

# ****10、switch 是否能作用在byte 上，是否能作用在long 上，是否能作用在String上？****

expr可以是byte、short、char、int、enum、String类型，但是long类型不能

# ****11、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）。

# ****12、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。

# ****13、构造器（constructor）是否可被重写（override）？****

答：构造器不能被继承，因此不能被重写，但可以被重载。

# ****14、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

# ****15、是否可以继承String类？****

答：String 类是final类，不可以被继承。

# ****16、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

# ****17、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/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。

# ****18、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。

# ****19、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。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

# ****20、抽象类（abstract class）和接口（interface）有什么异同？****

答：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。

# ****21、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，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

# ****22、GC是什么？为什么要有GC？****

答：GC是垃圾收集的意思，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

# ****23、String s = new String(“xyz”);创建了几个字符串对象？****

答：两个对象，一个是静态区的”xyz”，一个是用new创建在堆上的对象。

# ****24、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，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。

# ****25、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）。

# ****26、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。

**27、数据类型之间的转换：  
- 如何将字符串转换为基本数据类型？  
- 如何将基本数据类型转换为字符串？**  
答：  
- 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；

- 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

# ****28、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);

  }

# ****29、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 = **new** String(s1.getBytes("GB2312"), "ISO-8859-1");

# ****30、利用java.text.DataFormat 的子类（如SimpleDateFormat类）中的format(Date)方法可将日期格式化****

**class** DateFormatTest {

**public** **static** **void** main(String[] args) {

        SimpleDateFormat oldFormatter = **new** SimpleDateFormat("yyyy/MM/dd");

        Date date1 = **new** Date();

        System.out.println(oldFormatter.format(date1));

    }

}

# ****31、比较一下Java和JavaSciprt。****

**java是静态语言，js是动态语言**

- 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。  
- 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。（目前的浏览器几乎都使用了JIT（即时编译）技术来提升JavaScript的运行效率）  
- 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中变量是弱类型的，甚至在使用变量前可以不作声明，JavaScript的解释器在运行时检查推断其数据类型。

- 代码格式不一样。

**32、try{}里有一个return语句，那么紧跟在这个try后的finally{}里的代码会不会被执行，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。

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，就会返回修改后的值

# ****33、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 （下标越界异常） - NullPointerException （空指针异常）

- SecurityException （安全异常）

**34、类ExampleA继承Exception，类ExampleB继承ExampleA。**  
有如下代码片断：

**try** {

**throw** **new** ExampleB("b")

} **catch**（ExampleA e）{

    System.out.println("ExampleA");

} **catch**（Exception e）{

    System.out.println("Exception");

}

**请问执行此段代码的输出是什么？**  
答：输出：ExampleA。（根据里氏代换原则[能使用父类型的地方一定能使用子类型]，抓取ExampleA类型异常的catch块能够抓住try块中抛出的ExampleB类型的异常）

**面试题 - 说出下面代码的运行结果。（此题的出处是《Java编程思想》一书）**

**class** Annoyance **extends** Exception {}

**class** Sneeze **extends** Annoyance {}

**class** Human {

**public** **static** **void** main(String[] args)

**throws** Exception {

**try** {

**try** {

**throw** **new** Sneeze();

            }

**catch** ( Annoyance a ) {

                System.out.println("Caught Annoyance");

**throw** a;

            }

        }

**catch** ( Sneeze s ) {

            System.out.println("Caught Sneeze");

**return** ;

        }

**finally** {

            System.out.println("Hello World!");

        }

    }

}

**输出：**

Caught Annoyance

 Caught Sneeze

Hello World!

**[java]** [view plain](http://csdnimg.cn/release/phoenix/) [copy](http://csdnimg.cn/release/phoenix/)

1. **try** {
2. **throw** **new** Annoyance();
3. } **catch** (Sneeze s) {
4. System.out.println("Caught Sneeze");
5. **return**;
6. } **finally** {
7. System.out.println("Hello World!");
8. }

**输出：（父类throw出来的异常，子类并没有捕获到）**

Hello World!    
Exception in thread "main" com.xq.exceptions.Annoyance

    at com.xq.exceptions.Human.main(ExceptionTest.java:14)

**[java]** [view plain](http://csdnimg.cn/release/phoenix/) [copy](http://csdnimg.cn/release/phoenix/)

1. **try** {
2. **throw** **new** Annoyance();
3. } **catch** (Sneeze s) {
4. System.out.println("Caught Sneeze");
5. **return**;
6. } **catch** (Exception e) {
7. System.out.println("Caught Exception");
8. **return**;
9. } **finally** {
10. System.out.println("Hello World!");
11. }

**输出：（既然子类捕获不了，那就使用Exception），可以看到结果如下：**

Caught Exception

Hello World!

# ****35、List、Set、Map是否继承自Collection接口？****

答：List、Set 是，Map 不是。Map是键值对映射容器，与List和Set有明显的区别，而Set存储的零散的元素且不允许有重复元素（数学中的集合也是如此），List是线性结构的容器，适用于按数值索引访问元素的情形。

# ****36、Collection和Collections的区别？****

答：Collection是一个接口，它是Set、List等容器的父接口；Collections是个一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。

# ****37、List、Map、Set三个接口存取元素时，各有什么特点？****

答：List以特定索引来存取元素，可以有重复元素。Set不能存放重复元素（用对象的equals()方法来区分元素是否重复）。Map保存键值对（key-value pair）映射，映射关系可以是一对一或多对一

# ****38、Thread类的sleep()方法和对象的wait()方法都可以让线程暂停执行，它们有什么区别?****

答：sleep()方法（休眠）是线程类（Thread）的静态方法，调用此方法会让当前线程暂停执行指定的时间，将执行机会（CPU）让给其他线程，但是对象的锁依然保持，因此休眠时间结束后会自动恢复。wait()是Object类的方法，调用对象的wait()方法导致当前线程放弃对象的锁（线程暂停执行），进入对象的等待池（wait pool），只有调用对象的notify()方法（或notifyAll()方法）时才能唤醒等待池中的线程进入等锁池（lock pool），如果线程重新获得对象的锁就可以进入就绪状态。

# ****39、线程的sleep()方法和yield()方法有什么区别？****

答：  
① sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会；  
② 线程执行sleep()方法后转入阻塞（blocked）状态，而执行yield()方法后转入就绪（ready）状态；  
③ sleep()方法声明抛出InterruptedException，而yield()方法没有声明任何异常；

④ sleep()方法比yield()方法（跟操作系统CPU调度相关）具有更好的可移植性。

# ****40、当一个线程进入一个对象的synchronized方法A之后，其它线程是否可进入此对象的synchronized方法B？****

答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。因为非静态方法上的synchronized修饰符要求执行方法时要获得对象的锁，如果已经进入A方法说明对象锁已经被取走，那么试图进入B方法的线程就只能在等锁池（注意不是等待池哦）中等待**对象的锁。**

# ****41、请说出与线程同步以及线程调度相关的方法。****

答：  
- wait()：使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；  
- sleep()：使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要处理InterruptedException异常；  
- notify()：唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且与优先级无关；  
- notityAll()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

# ****42、编写多线程程序有几种实现方式？****

答：一种是继承Thread类；另一种是实现Runnable接口。两种方式都要通过重写run()方法来定义线程的行为，推荐使用后者，因为Java中的继承是单继承，一个类有一个父类，如果继承了Thread类就无法再继承其他类了，显然使用Runnable接口更为灵活。

# ****43、synchronized关键字的用法？****

答：synchronized关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用synchronized(对象) { … }定义同步代码块，或者在声明方法时将synchronized作为方法的修饰符。

# ****44、举例说明同步和异步。****

答：如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（数据库操作中的排他锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

# ****45、简述synchronized 和java.util.concurrent.locks.Lock的异同？****

答：Lock是Java 5以后引入的新的API，和关键字synchronized相比主要相同点：Lock 能完成synchronized所实现的所有功能；主要不同点：Lock有比synchronized更精确的线程语义和更好的性能，而且不强制性的要求一定要获得锁。synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且最好在finally 块中释放（这是释放外部资源的最好的地方）

**46、事务的ACID是指什么？**

答：  
- 原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；  
- 一致性(Consistent)：事务结束后系统状态是一致的；- 隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；  
- 持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

# ****47、获得一个类的类对象有哪些方式？****

答：  
- 方法1：类型.class，例如：String.class  
- 方法2：对象.getClass()，例如：”hello”.getClass()

- 方法3：Class.forName()，例如：Class.forName(“java.lang.String”)

# ****48、简述一下面向对象的”六原则一法则”。****

答：

**- 单一职责原则：**一个类只做它该做的事情。（单一职责原则想表达的就是”高内聚”，写代码最终极的原则只有六个字”高内聚、低耦合”，就如同葵花宝典或辟邪剑谱的中心思想就八个字”欲练此功必先自宫”，所谓的高内聚就是一个代码模块只完成一项功能，在面向对象中，如果只让一个类完成它该做的事，而不涉及与它无关的领域就是践行了高内聚的原则，这个类就只有单一职责。我们都知道一句话叫”因为专注，所以专业”，一个对象如果承担太多的职责，那么注定它什么都做不好。这个世界上任何好的东西都有两个特征，一个是功能单一，好的相机绝对不是电视购物里面卖的那种一个机器有一百多种功能的，它基本上只能照相；另一个是模块化，好的自行车是组装车，从减震叉、刹车到变速器，所有的部件都是可以拆卸和重新组装的，好的乒乓球拍也不是成品拍，一定是底板和胶皮可以拆分和自行组装的，一个好的软件系统，它里面的每个功能模块也应该是可以轻易的拿到其他系统中使用的，这样才能实现软件复用的目标。）

**- 开闭原则：**软件实体应当对扩展开放，对修改关闭。（在理想的状态下，当我们需要为一个软件系统增加新功能时，只需要从原来的系统派生出一些新类就可以，不需要修改原来的任何一行代码。要做到开闭有两个要点：①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点；②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂

**- 依赖倒转原则：**面向接口编程。

**- 里氏替换原则：**任何时候都可以用子类型替换掉父类型。（关于里氏替换原则的描述，Barbara Liskov女士的描述比这个要复杂得多，但简单的说就是能用父类型的地方就一定能使用子类型。里氏替换原则可以检查继承关系是否合理，如果一个继承关系违背了里氏替换原则，那么这个继承关系一定是错误的，需要对代码进行重构。例如让猫继承狗，或者狗继承猫，又或者让正方形继承长方形都是错误的继承关系，因为你很容易找到违反里氏替换原则的场景。需要注意的是：子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。）

**- 接口隔离原则：**接口要小而专，绝不能大而全。（臃肿的接口是对接口的污染，既然接口表示能力，那么一个接口只应该描述一种能力，接口也应该是高度内聚的。例如，琴棋书画就应该分别设计为四个接口，而不应设计成一个接口中的四个方法，因为如果设计成一个接口中的四个方法，那么这个接口很难用，毕竟琴棋书画四样都精通的人还是少数，而如果设计成四个接口，会几项就实现几个接口，这样的话每个接口被复用的可能性是很高的。Java中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。）

**- 合成聚合复用原则：**优先使用聚合或合成关系复用代码。

**- 迪米特法则：**迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。（迪米特法则简单的说就是如何做到”低耦合”，门面模式和调停者模式就是对迪米特法则的践行。对于门面模式可以举一个简单的例子，你去一家公司洽谈业务，你不需要了解这个公司内部是如何运作的，你甚至可以对这个公司一无所知，去的时候只需要找到公司入口处的前台美女，告诉她们你要做什么，她们会找到合适的人跟你接洽，前台的美女就是公司这个系统的门面。再复杂的系统都可以为用户提供一个简单的门面，Java Web开发中作为前端控制器的Servlet或Filter不就是一个门面吗，浏览器对服务器的运作方式一无所知，但是通过前端控制器就能够根据你的请求得到相应的服务。调停者模式也可以举一个简单的例子来说明，例如一台计算机，CPU、内存、硬盘、显卡、声卡各种设备需要相互配合才能很好的工作，但是如果这些东西都直接连接到一起，计算机的布线将异常复杂，在这种情况下，主板作为一个调停者的身份出现，它将各个设备连接在一起而不需要每个设备之间直接交换数据，这样就减小了系统的耦合度和复杂度，如下图所示。迪米特法则用通俗的话来将就是不要和陌生人打交道，如果真的需要，找一个自己的朋友，让他替你和陌生人打交道。）

# ****49、简述一下你了解的设计模式。****

答：所谓设计模式，就是一套被反复使用的代码设计经验的总结（情境中一个问题经过证实的一个解决方案）。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性

**几个常用的设计模式:**

**- 工厂模式：**工厂类可以根据条件生成不同的子类实例，这些子类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作（多态方法）。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

**- 代理模式：**给一个对象提供一个代理对象，并由代理对象控制原对象的引用。

**- 适配器模式：**把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起使用的类能够一起工作。

**- 单例模式：**一个类只有一个实例，即一个类只有一个对象实例。

懒汉式单例模式，线程不安全,致命的是在多线程不能正常工作

**public** **class** Singleton {

**private** **static** Singleton instance = **null**;

**private** Singleton() {}

**public** **static** **synchronized** Singleton getInstance(){

**if** (instance == **null**) instance ＝ **new** Singleton();

**return** instance;

    }

}

饿汉式单例模式,避免了多线程的同步问题

**public** **class** Singleton {

**private** Singleton(){}

**private** **static** Singleton instance = **new** Singleton();

**public** **static** Singleton getInstance(){

**return** instance;

    }

}

# ****50、用Java写一个冒泡排序。****

for(int i=0;i<arr.length-1;i++){//外层循环控制排序趟数

　　　　　　for(int j=0;j<arr.length-1-i;j++){//内层循环控制每一趟排序多少次

　　　　　　　　if(arr[j]>arr[j+1]){

　　　　　　　　　　int temp=arr[j];

　　　　　　　　　　arr[j]=arr[j+1];

　　　　　　　　　　arr[j+1]=temp;

　　　　　　　　}

　　　　　　}

　　　　}

# **51、用Java写一个二分查找。**

非递归实现：

public static int biSearch(int []array,int a){

int lo=0;

int hi=array.length-1;

int mid;

while(lo<=hi){

mid=(lo+hi)/2;

if(array[mid]==a){

return mid+1;

}else if(array[mid]<a){

lo=mid+1;

}else{

hi=mid-1;

}

}

return -1;

}

递归实现：

public static int sort(int []array,int a,int lo,int hi){

if(lo<=hi){

int mid=(lo+hi)/2;

if(a==array[mid]){

return mid+1;

}

else if(a>array[mid]){

return sort(array,a,mid+1,hi);

}else{

return sort(array,a,lo,mid-1);

}

}

return -1;

}

# ****52、Servlet的运行过程？****

Web容器加载Servlet并将其实例化后，Servlet生命周期开始，容器运行其init()方法进行Servlet的初始化；请求到达时调用Servlet的service()方法，service()方法会根据需要调用与请求对应的doGet或doPost等方法；当服务器关闭或项目被卸载时服务器会将Servlet实例销毁，此时会调用Servlet的destroy()方法。

# ****53、转发（forward）和重定向（redirect）的区别？****

答：forward是容器中控制权的转向，是服务器请求资源，服务器直接访问目标地址的URL，把那个URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。redirect就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址，很明显redirect无法访问到服务器保护起来资源，但是可以从一个网站redirect到其他网站。forward更加高效，所以在满足需要时尽量使用forward（通过调用RequestDispatcher对象的forward()方法，该对象可以通过ServletRequest对象的getRequestDispatcher()方法获得），并且这样也有助于隐藏实际的链接；在有些情况下，比如需要访问一个其它服务器上的资源，则必须使用重定向（通过HttpServletResponse对象调用其sendRedirect()方法实现）。

# ****54、JSP有哪些内置对象？作用分别是什么？****

答：JSP有9个内置对象：   
- request：封装客户端的请求，其中包含来自GET或POST请求的参数；   
- response：封装服务器对客户端的响应；   
- pageContext：通过该对象可以获取其他对象；   
- session：封装用户会话的对象；   
- application：封装服务器运行环境的对象；   
- out：输出服务器响应的输出流对象；   
- config：Web应用的配置对象；   
- page：JSP页面本身（相当于Java程序中的this）；   
- exception：封装页面抛出异常的对象。

# ****55、讲解JSP中的四种作用域。****

答：JSP中的四种作用域包括page、request、session和application，具体来说：   
- page代表与一个页面相关的对象和属性。   
- request代表与Web客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个Web组件；需要在页面显示的临时数据可以置于此作用域。   
- session代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的session中。

- application代表与整个Web应用程序相关的对象和属性，它实质上是跨越整个Web应用程序，包括多个页面、请求和会话的一个全局作用域。