## ****java常见知识点总结：****

## 面向对象和面向过程的区别

面向过程：

**优点**：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源;比如单片机、嵌入式开发、Linux/Unix等一般采用面向过程开发，性能是最重要的因素。

**缺点**：没有面向对象易维护、易复用、易扩展

面向对象：

**优点**： 易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护

**缺点**： 性能比面向过程低

## java语言有哪些特点

1、简单易学；

2、面向对象（封装，继承，多态）；

3、平台无关性（ Java 虚拟机实现平台无关性）；

4、可靠性；

5、安全性；

6、支持多线程（ C++ 语言没有内置的多线程机制，因此必须调用操作系统的多线程功能来进行多线程程序设计，而 Java 语言却提供了多线程支持）；

7、支持网络编程并且很方便（ Java 语言诞生本身就是为简化网络编程设计的，因此 Java 语言不仅支持网络编程而且很方便）；

8、编译与解释并存

## 3、什么是jdk，什么是jre，什么是jvm，三者之间的联系和区别

这几个是Java中很基本很基本的东西，但是我相信一定还有很多人搞不清楚！为什么呢？因为我们大多数时候在使用现成的编译工具以及环境的时候，并没有去考虑这些东西。

**JDK:** 顾名思义它是给开发者提供的开发工具箱,是给程序开发者用的。它除了包括完整的JRE（Java Runtime Environment），Java运行环境，还包含了其他供开发者使用的工具包。

**JRE:** 普通用户而只需要安装 JRE（Java Runtime Environment）来运行 Java 程序。而程序开发者必须安装JDK来编译、调试程序。

**JVM：** 当我们运行一个程序时，JVM 负责将字节码转换为特定机器代码，JVM 提供了内存管理/垃圾回收和安全机制等。这种独立于硬件和操作系统，正是 java 程序可以一次编写多处执行的原因。

**区别与联系：**

1. JDK 用于开发，JRE 用于运行java程序 ；
2. JDK 和 JRE 中都包含 JVM ；
3. JVM 是 java 编程语言的核心并且具有平台独立性。

## 4、什么是字节码，采用字节码的最大好处是什么？

**先看下 java 中的编译器和解释器：**

Java 中引入了虚拟机的概念，即在机器和编译程序之间加入了一层抽象的虚拟的机器。这台虚拟的机器在任何平台上都提供给编译程序一个的共同的接口。

编译程序只需要面向虚拟机，生成虚拟机能够理解的代码，然后由解释器来将虚拟机代码转换为特定系统的机器码执行。在 Java 中，这种供虚拟机理解的代码叫做字节码（即扩展名为 .class 的文件），它不面向任何特定的处理器，只面向虚拟机。

每一种平台的解释器是不同的，但是实现的虚拟机是相同的。Java 源程序经过编译器编译后变成字节码，字节码由虚拟机解释执行，虚拟机将每一条要执行的字节码送给解释器，解释器将其翻译成特定机器上的机器码，然后在特定的机器上运行。这也就是解释了 Java 的编译与解释并存的特点。

Java 源代码---->编译器---->jvm 可执行的 Java 字节码(即虚拟指令)---->jvm---->jvm 中解释器----->机器可执行的二进制机器码---->程序运行。

**采用字节码的好处：**

**Java 语言通过字节码的方式，在一定程度上解决了传统解释型语言执行效率低的问题，同时又保留了解释型语言可移植的特点。所以 Java 程序运行时比较高效，而且，由于字节码并不专对一种特定的机器，因此，Java程序无须重新编译便可在多种不同的计算机上运行。**

解释性语言：解释型语言，是在运行的时候将程序翻译成机器语言。解释型语言的程序不需要在运行前编译，在运行程序的时候才翻译，专门的解释器负责在每个语句执行的时候解释程序代码。这样解释型语言每执行一次就要翻译一次，效率比较低。——百度百科

## java和C++的区别

我知道很多人没学过 C++，但是面试官就是没事喜欢拿咱们 Java 和 C++ 比呀！没办法！！！就算没学过C++，也要记下来！

* 都是面向对象的语言，都支持封装、继承和多态
* Java 不提供指针来直接访问内存，程序内存更加安全
* Java 的类是单继承的，C++ 支持多重继承；虽然 Java 的类不可以多继承，但是接口可以多继承。
* Java 有自动内存管理机制，不需要程序员手动释放无用内存

## 什么是java程序的主类，应用程序和java小程序的主类有何不同？

一个程序中可以有多个类，但只能有一个类是主类。在 Java 应用程序中，这个主类是指包含 main（）方法的类。而在 Java 小程序中，这个主类是一个继承自系统类 JApplet 或 Applet 的子类。应用程序的主类不一定要求是 public 类，但小程序的主类要求必须是 public 类。主类是 Java 程序执行的入口点。

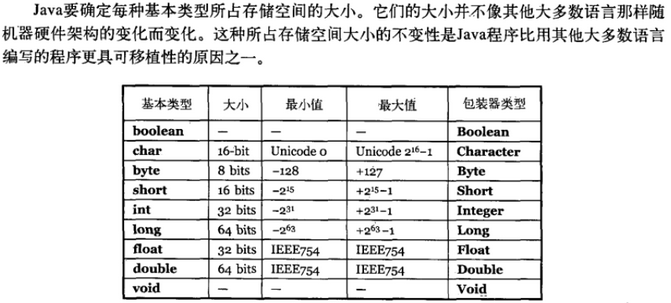
## 7、Java 应用程序与小程序之间有那些差别

简单说应用程序是从主线程启动(也就是 main() 方法)。applet 小程序没有main方法，主要是嵌在浏览器页面上运行(调用init()线程或者run()来启动)，嵌入浏览器这点跟 flash 的小游戏类似。

## 8、字符型常量和字符串常量的区别

1. 形式上: 字符常量是单引号引起的一个字符 字符串常量是双引号引起的若干个字符
2. 含义上: 字符常量相当于一个整形值( ASCII 值),可以参加表达式运算 字符串常量代表一个地址值(该字符串在内存中存放位置)
3. 占内存大小 字符常量只占2个字节 字符串常量占若干个字节(至少一个字符结束标志) (**注意： char在Java中占两个字节**)

java编程思想第四版：2.2.2节



## 构造器 Constructor 是否可被 override

在讲继承的时候我们就知道父类的私有属性和构造方法并不能被继承，所以 Constructor 也就不能被 override（重写）,但是可以 overload（重载）,所以你可以看到一个类中有多个构造函数的情况。

## 10、重载和重写的区别

**重载：** 发生在同一个类中，方法名必须相同，参数类型不同、个数不同、顺序不同，方法返回值和访问修饰符可以不同，发生在编译时。

**重写：** 发生在父子类中，方法名、参数列表必须相同，返回值范围小于等于父类，抛出的异常范围小于等于父类，访问修饰符范围大于等于父类；如果父类方法访问修饰符为 private 则子类就不能重写该方法。

## 11、Java 面向对象编程三大特性:封装、继承、多态

**封装**

封装把一个对象的属性私有化，同时提供一些可以被外界访问的属性的方法，如果属性不想被外界访问，我们大可不必提供方法给外界访问。但是如果一个类没有提供给外界访问的方法，那么这个类也没有什么意义了。

**继承**

继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。通过使用继承我们能够非常方便地复用以前的代码。

**关于继承如下 3 点请记住：**

1. 子类拥有父类非 private 的属性和方法。
2. 子类可以拥有自己属性和方法，即子类可以对父类进行扩展。
3. 子类可以用自己的方式实现父类的方法。（以后介绍）。

**多态**

所谓多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。

在Java中有两种形式可以实现多态：继承（多个子类对同一方法的重写）和接口（实现接口并覆盖接口中同一方法）。

## 12、String 和 StringBuffer、StringBuilder 的区别是什么,String 为什么是不可变的?

****可变性****

简单的来说：String 类中使用 final 关键字字符数组保存字符串，private　final　char　value[]，所以 String 对象是不可变的。而StringBuilder 与 StringBuffer 都继承自 AbstractStringBuilder 类，在 AbstractStringBuilder 中也是使用字符数组保存字符串char[]value 但是没有用 final 关键字修饰，所以这两种对象都是可变的。

StringBuilder 与 StringBuffer 的构造方法都是调用父类构造方法也就是 AbstractStringBuilder 实现的，大家可以自行查阅源码。

AbstractStringBuilder.java

abstract class AbstractStringBuilder implements Appendable, CharSequence {

char[] value;

int count;

AbstractStringBuilder() {

}

AbstractStringBuilder(int capacity) {

value = new char[capacity];

}

**线程安全性**

String 中的对象是不可变的，也就可以理解为常量，线程安全。AbstractStringBuilder 是 StringBuilder 与 StringBuffer 的公共父类，定义了一些字符串的基本操作，如 expandCapacity、append、insert、indexOf 等公共方法。StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的。StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

**性能**

每次对 String 类型进行改变的时候，都会生成一个新的 String 对象，然后将指针指向新的 String 对象。StringBuffer 每次都会对 StringBuffer 对象本身进行操作，而不是生成新的对象并改变对象引用。相同情况下使用 StirngBuilder 相比使用 StringBuffer 仅能获得 10%~15% 左右的性能提升，但却要冒多线程不安全的风险。

**对于三者使用的总结：**

1. 操作少量的数据 = String
2. 单线程操作字符串缓冲区下操作大量数据 = StringBuilder
3. 多线程操作字符串缓冲区下操作大量数据 = StringBuffer

## 13、自动装箱与拆箱

**装箱**：将基本类型用它们对应的引用类型包装起来；

**拆箱**：将包装类型转换为基本数据类型；

## 14、在一个静态方法内调用一个非静态成员为什么是非法的

由于静态方法可以不通过对象进行调用，因此在静态方法里，不能调用其他非静态变量，也不可以访问非静态变量成员。

## 15、在 Java 中定义一个不做事且没有参数的构造方法的作用

Java 程序在执行子类的构造方法之前，如果没有用 super() 来调用父类特定的构造方法，则会调用父类中“没有参数的构造方法”。因此，如果父类中只定义了有参数的构造方法，而在子类的构造方法中又没有用 super() 来调用父类中特定的构造方法，则编译时将发生错误，因为 Java 程序在父类中找不到没有参数的构造方法可供执行。解决办法是在父类里加上一个不做事且没有参数的构造方法。

## 16、import java和javax有什么区别

刚开始的时候 JavaAPI 所必需的包是 java 开头的包，javax 当时只是扩展 API 包来说使用。然而随着时间的推移，javax 逐渐的扩展成为 Java API 的组成部分。但是，将扩展从 javax 包移动到 java 包将是太麻烦了，最终会破坏一堆现有的代码。因此，最终决定 javax 包将成为标准API的一部分。

所以，实际上java和javax没有区别。这都是一个名字。

## 17、接口和抽象类的区别是什么

1. 接口的方法默认是 public，所有方法在接口中不能有实现(Java 8 开始接口方法可以有默认实现），抽象类可以有非抽象的方法
2. 接口中的实例变量默认是 final 类型的，而抽象类中则不一定
3. 一个类可以实现多个接口，但最多只能实现一个抽象类
4. 一个类实现接口的话要实现接口的所有方法，而抽象类不一定
5. 接口不能用 new 实例化，但可以声明，但是必须引用一个实现该接口的对象 从设计层面来说，抽象是对类的抽象，是一种模板设计，接口是行为的抽象，是一种行为的规范。

## 18、成员变量与局部变量的区别有那些

1. 从语法形式上，看成员变量是属于类的，而局部变量是在方法中定义的变量或是方法的参数；成员变量可以被 public,private,static 等修饰符所修饰，而局部变量不能被访问控制修饰符及 static 所修饰；但是，成员变量和局部变量都能被 final 所修饰；
2. 从变量在内存中的存储方式来看，成员变量是对象的一部分，而对象存在于堆内存，局部变量存在于栈内存
3. 从变量在内存中的生存时间上看，成员变量是对象的一部分，它随着对象的创建而存在，而局部变量随着方法的调用而自动消失。
4. 成员变量如果没有被赋初值，则会自动以类型的默认值而赋值（一种情况例外被 final 修饰但没有被 static 修饰的成员变量必须显示地赋值）；而局部变量则不会自动赋值。

## 19、创建一个对象用什么运算符？对象实体与对象引用有何不同？

new运算符，new创建对象实例（对象实例在堆内存中），对象引用指向对象实例（对象引用存放在栈内存中）。一个对象引用可以指向0个或1个对象（一根绳子可以不系气球，也可以系一个气球）;一个对象可以有n个引用指向它（可以用n条绳子系住一个气球）。

## 20、什么是方法的返回值？返回值在类的方法里的作用是什么？

方法的返回值是指我们获取到的某个方法体中的代码执行后产生的结果！（前提是该方法可能产生结果）。返回值的作用:接收出结果，使得它可以用于其他的操作！

## 21、一个类的构造方法的作用是什么 若一个类没有声明构造方法，该程序能正确执行吗 为什么

主要作用是完成对类对象的初始化工作。可以执行。因为一个类即使没有声明构造方法也会有默认的不带参数的构造方法。

## 22、构造方法有哪些特性

1. 名字与类名相同；
2. 没有返回值，但不能用void声明构造函数；
3. 生成类的对象时自动执行，无需调用。

## 23、静态方法和实例方法有何不同

1、在外部调用静态方法时，可以使用"类名.方法名"的方式，也可以使用"对象名.方法名"的方式。而实例方法只有后面这种方式。也就是说，调用静态方法可以无需创建对象。

2、静态方法在访问本类的成员时，只允许访问静态成员（即静态成员变量和静态方法），而不允许访问实例成员变量和实例方法；实例方法则无此限制。

## 24、对象的相等与指向他们的引用相等，两者有什么不同？

对象的相等，比的是内存中存放的内容是否相等。而引用相等，比较的是他们指向的内存地址是否相等。

## 25、在调用子类构造方法之前会先调用父类没有参数的构造方法，其目的是？

帮助子类做初始化工作。

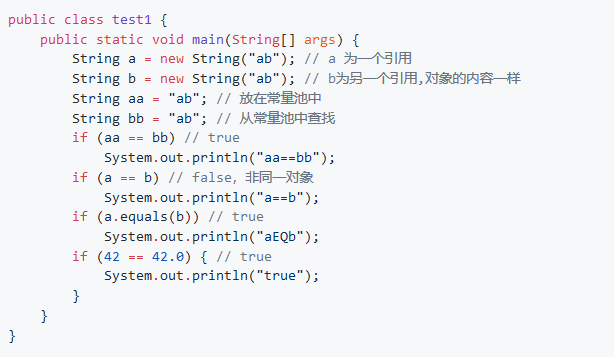
## 26、== 与 equals(重要)

**==** : 它的作用是判断两个对象的地址是不是相等。即，判断两个对象是不是同一个对象。(基本数据类型==比较的是值，引用数据类型==比较的是内存地址)

**equals()** : 它的作用也是判断两个对象是否相等。但它一般有两种使用情况：

* 情况1：类没有覆盖 equals() 方法。则通过 equals() 比较该类的两个对象时，等价于通过“==”比较这两个对象。
* 情况2：类覆盖了 equals() 方法。一般，我们都覆盖 equals() 方法来两个对象的内容相等；若它们的内容相等，则返回 true (即，认为这两个对象相等)。

**举个例子：**



**说明：**

* String 中的 equals 方法是被重写过的，因为 object 的 equals 方法是比较的对象的内存地址，而 String 的 equals 方法比较的是对象的值。
* 当创建 String 类型的对象时，虚拟机会在常量池中查找有没有已经存在的值和要创建的值相同的对象，如果有就把它赋给当前引用。如果没有就在常量池中重新创建一个 String 对象。

## 27、hashCode 与 equals（重要）

面试官可能会问你：“你重写过 hashcode 和 equals 么，为什么重写equals时必须重写hashCode方法？”

**hashCode（）介绍**

hashCode() 的作用是获取哈希码，也称为散列码；它实际上是返回一个int整数。这个哈希码的作用是确定该对象在哈希表中的索引位置。hashCode() 定义在JDK的Object.java中，这就意味着Java中的任何类都包含有hashCode() 函数。

散列表存储的是键值对(key-value)，它的特点是：能根据“键”快速的检索出对应的“值”。这其中就利用到了散列码！（可以快速找到所需要的对象）

**为什么要有 hashCode**

**我们以“HashSet 如何检查重复”为例子来说明为什么要有 hashCode：**

当你把对象加入 HashSet 时，HashSet 会先计算对象的 hashcode 值来判断对象加入的位置，同时也会与其他已经加入的对象的 hashcode 值作比较，如果没有相符的hashcode，HashSet会假设对象没有重复出现。但是如果发现有相同 hashcode 值的对象，这时会调用 equals（）方法来检查 hashcode 相等的对象是否真的相同。如果两者相同，HashSet 就不会让其加入操作成功。如果不同的话，就会重新散列到其他位置。（摘自我的Java启蒙书《Head fist java》第二版）。这样我们就大大减少了 equals 的次数，相应就大大提高了执行速度。

**为什么两个对象有相同的hashcode值，它们也不一定是相等的？**

在这里解释一位小伙伴的问题。以下内容摘自《Head Fisrt Java》。

因为hashCode() 所使用的杂凑算法也许刚好会让多个对象传回相同的杂凑值。越糟糕的杂凑算法越容易碰撞，但这也与数据值域分布的特性有关（所谓碰撞也就是指的是不同的对象得到相同的 hashCode）。

我们刚刚也提到了 HashSet,如果 HashSet 在对比的时候，同样的 hashcode 有多个对象，它会使用 equals() 来判断是否真的相同。也就是说 hashcode 只是用来缩小查找成本。

**hashCode（）与equals（）的相关规定**

1. 如果两个对象相等，则hashcode一定也是相同的
2. 两个对象相等,对两个对象分别调用equals方法都返回true
3. 两个对象有相同的hashcode值，它们也不一定是相等的
4. **因此，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖**
5. hashCode() 的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

## 值调用和引用调用

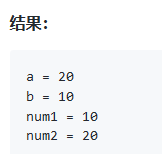
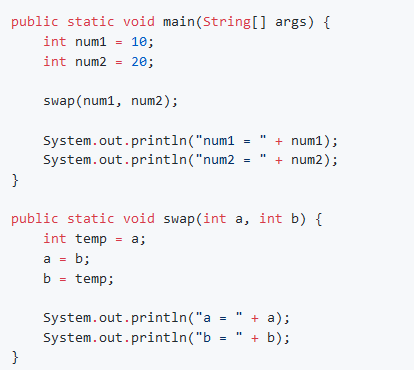
为什么 Java 中只有值传递？

首先回顾一下在程序设计语言中有关将参数传递给方法（或函数）的一些专业术语。按值调用(call by value)表示方法接收的是调用者提供的值，而按引用调用（call by reference)表示方法接收的是调用者提供的变量地址。一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。 它用来描述各种程序设计语言（不只是Java)中方法参数传递方式。

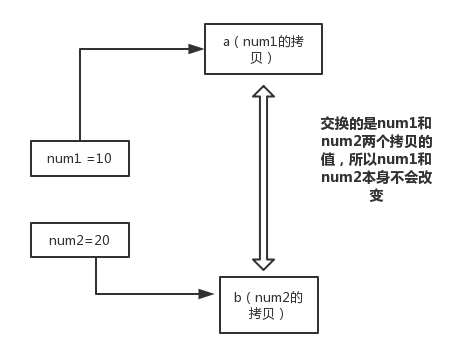
Java程序设计语言总是采用按值调用。也就是说，方法得到的是所有参数值的一个拷贝，也就是说，方法不能修改传递给它的任何参数变量的内容。

下面通过 3 个例子来给大家说明

example 1：



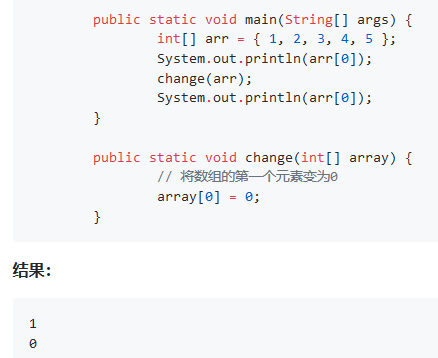
解析：

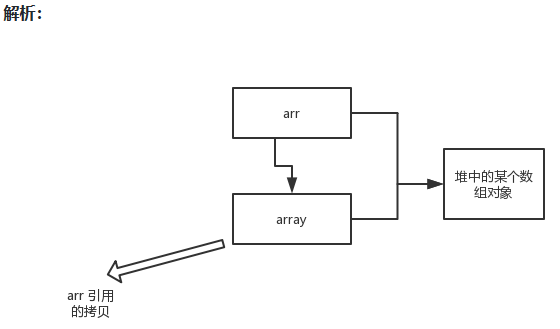


在swap方法中，a、b的值进行交换，并不会影响到 num1、num2。因为，a、b中的值，只是从 num1、num2 的复制过来的。也就是说，a、b相当于num1、num2 的副本，副本的内容无论怎么修改，都不会影响到原件本身。

通过上面例子，我们已经知道了一个方法不能修改一个基本数据类型的参数，而对象引用作为参数就不一样，请看 example2.

**example 2：**



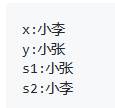
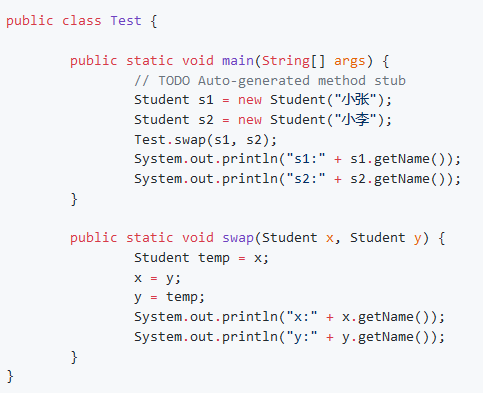


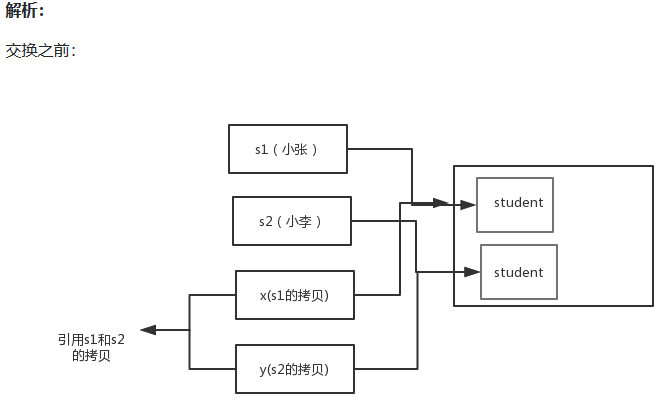
array 被初始化 arr 的拷贝也就是一个对象的引用，也就是说 array 和 arr 指向的时同一个数组对象。 因此，外部对引用对象的改变会反映到所对应的对象上。

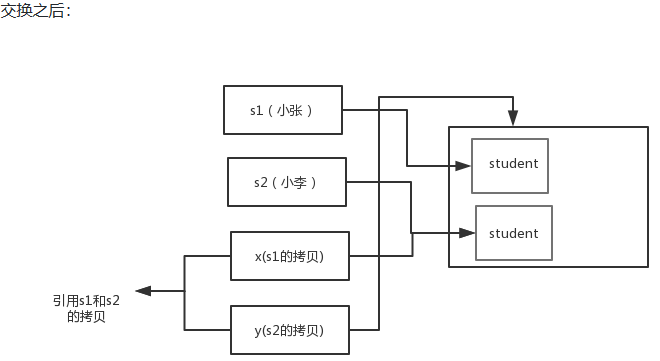
通过 example2 我们已经看到，实现一个改变对象参数状态的方法并不是一件难事。理由很简单，方法得到的是对象引用的拷贝，对象引用及其他的拷贝同时引用同一个对象。

很多程序设计语言（特别是，C++和Pascal)提供了两种参数传递的方式：值调用和引用调用。有些程序员（甚至本书的作者）认为Java程序设计语言对对象采用的是引用调用，实际上，这种理解是不对的。由于这种误解具有一定的普遍性，所以下面给出一个**反例**来详细地阐述一下这个问题。

**example 3**







通过上面两张图可以很清晰的看出： 方法并没有改变存储在变量 s1 和 s2 中的对象引用。swap方法的参数x和y被初始化为两个对象引用的拷贝，这个方法交换的是这两个拷贝

**总结：**

Java程序设计语言对对象采用的不是引用调用，实际上，对象引用是按值传递的。

下面再总结一下Java中方法参数的使用情况：

* 一个方法不能修改一个基本数据类型的参数（即数值型或布尔型）
* 一个方法可以改变一个对象参数的状态。
* 一个方法不能让对象参数引用一个新的对象。

## 29、简述线程，程序、进程的基本概念。以及他们之间关系是什么？

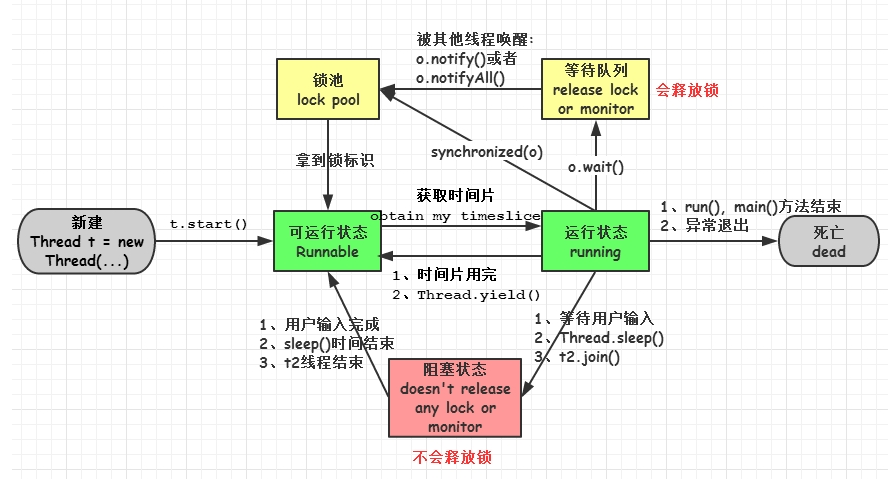
**线程**与进程相似，但线程是一个比进程更小的执行单位。一个进程在其执行的过程中可以产生多个线程。与进程不同的是同类的多个线程共享同一块内存空间和一组系统资源，所以系统在产生一个线程，或是在各个线程之间作切换工作时，负担要比进程小得多，也正因为如此，线程也被称为轻量级进程。

**程序**是含有指令和数据的文件，被存储在磁盘或其他的数据存储设备中，也就是说程序是静态的代码。

**进程**是程序的一次执行过程，是系统运行程序的基本单位，因此进程是动态的。系统运行一个程序即是一个进程从创建，运行到消亡的过程。简单来说，一个进程就是一个执行中的程序，它在计算机中一个指令接着一个指令地执行着，同时，每个进程还占有某些系统资源如CPU时间，内存空间，文件，文件，输入输出设备的使用权等等。换句话说，当程序在执行时，将会被操作系统载入内存中。 线程是进程划分成的更小的运行单位。线程和进程最大的不同在于基本上各进程是独立的，而各线程则不一定，因为同一进程中的线程极有可能会相互影响。从另一角度来说，进程属于操作系统的范畴，主要是同一段时间内，可以同时执行一个以上的程序，而线程则是在同一程序内几乎同时执行一个以上的程序段。

## 30、线程有哪些基本状态？这些状态是如何定义的?

1. **新建(new)**：新创建了一个线程对象。
2. **可运行(runnable)**：线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获 取cpu的使用权。
3. **运行(running)**：可运行状态(runnable)的线程获得了cpu时间片（timeslice），执行程序代码。
4. **阻塞(block)**：阻塞状态是指线程因为某种原因放弃了cpu使用权，也即让出了cpu timeslice，暂时停止运行。直到线程进入可运行(runnable)状态，才有 机会再次获得cpu timeslice转到运行(running)状态。阻塞的情况分三种： (一). 等待阻塞：运行(running)的线程执行o.wait()方法，JVM会把该线程放 入等待队列(waitting queue)中。 (二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则JVM会把该线程放入锁池(lock pool)中。 (三). 其他阻塞: 运行(running)的线程执行Thread.sleep(long ms)或t.join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入可运行(runnable)状态。
5. **死亡(dead)**：线程run()、main()方法执行结束，或者因异常退出了run()方法，则该线程结束生命周期。死亡的线程不可再次复生。



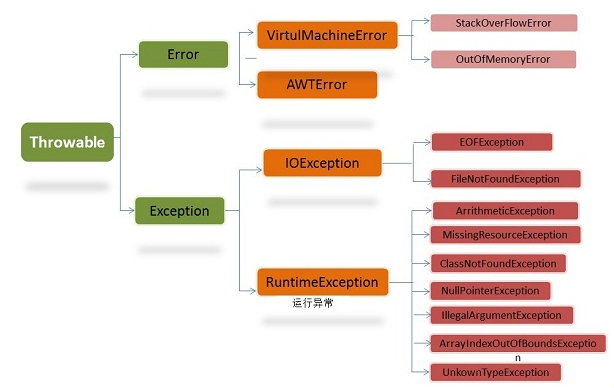
## 31、关于 final 关键字的一些总结

final关键字主要用在三个地方：变量、方法、类。

1. 对于一个final变量，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。
2. 当用final修饰一个类时，表明这个类不能被继承。final类中的所有成员方法都会被隐式地指定为final方法。
3. 使用final方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率。在早期的Java实现版本中，会将final方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升（现在的Java版本已经不需要使用final方法进行这些优化了）。类中所有的private方法都隐式地指定为fianl。

## 32、Java 中的异常处理

**Java异常类层次结构图**



在 Java 中，所有的异常都有一个共同的祖先java.lang包中的 **Throwable类**。Throwable： 有两个重要的子类：**Exception（异常）** 和 **Error（错误）** ，二者都是 Java 异常处理的重要子类，各自都包含大量子类。

**Error（错误）:是程序无法处理的错误**，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM（Java 虚拟机）出现的问题。例如，Java虚拟机运行错误（Virtual MachineError），当 JVM 不再有继续执行操作所需的内存资源时，将出现 OutOfMemoryError。这些异常发生时，Java虚拟机（JVM）一般会选择线程终止。

这些错误表示故障发生于虚拟机自身、或者发生在虚拟机试图执行应用时，如Java虚拟机运行错误（Virtual MachineError）、类定义错误（NoClassDefFoundError）等。这些错误是不可查的，因为它们在应用程序的控制和处理能力之外，而且绝大多数是程序运行时不允许出现的状况。对于设计合理的应用程序来说，即使确实发生了错误，本质上也不应该试图去处理它所引起的异常状况。在 Java中，错误通过Error的子类描述。

**Exception（异常）:是程序本身可以处理的异常**。Exception 类有一个重要的子类 **RuntimeException**。RuntimeException 异常由Java虚拟机抛出。**NullPointerException**（要访问的变量没有引用任何对象时，抛出该异常）、**ArithmeticException**（算术运算异常，一个整数除以0时，抛出该异常）和 **ArrayIndexOutOfBoundsException** （下标越界异常）。

**注意：异常和错误的区别：异常能被程序本身可以处理，错误是无法处理。**

**Throwable类常用方法**

* **public string getMessage()**:返回异常发生时的详细信息
* **public string toString()**:返回异常发生时的简要描述
* **public string getLocalizedMessage()**:返回异常对象的本地化信息。使用Throwable的子类覆盖这个方法，可以声称本地化信息。如果子类没有覆盖该方法，则该方法返回的信息与getMessage（）返回的结果相同
* **public void printStackTrace()**:在控制台上打印Throwable对象封装的异常信息

**异常处理总结**

* try 块：用于捕获异常。其后可接零个或多个catch块，如果没有catch块，则必须跟一个finally块。
* catch 块：用于处理try捕获到的异常。
* finally 块：无论是否捕获或处理异常，finally块里的语句都会被执行。当在try块或catch块中遇到return语句时，finally语句块将在方法返回之前被执行。

**在以下4种特殊情况下，finally块不会被执行：**

1. 在finally语句块中发生了异常。
2. 在前面的代码中用了System.exit()退出程序。
3. 程序所在的线程死亡。
4. 关闭CPU。

## 33、Java序列化中如果有些字段不想进行序列化怎么办？

对于不想进行序列化的变量，使用transient关键字修饰。

transient关键字的作用是：阻止实例中那些用此关键字修饰的变量序列化；当对象被反序列化时，被transient修饰的变量值不会被持久化和恢复。transient只能修饰变量，不能修饰类和方法。

## final，static，this，super关键字

**final 关键字**

final关键字主要用在三个地方：变量、方法、类。

1、对于一个final变量，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。

2、当用final修饰一个类时，表明这个类不能被继承。final类中的所有成员方法都会被隐式地指定为final方法。

3、使用final方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率。在早期的Java实现版本中，会将final方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升（现在的Java版本已经不需要使用final方法进行这些优化了）。类中所有的private方法都隐式地指定为final。

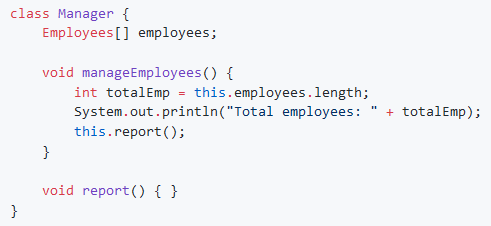
**static 关键字**

**static 关键字主要有以下四种使用场景：**

1. **修饰成员变量和成员方法:** 被 static 修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享，可以并且建议通过类名调用。被static 声明的成员变量属于静态成员变量，静态变量 存放在 Java 内存区域的方法区。调用格式：类名.静态变量名 类名.静态方法名()
2. **静态代码块:** 静态代码块定义在类中方法外, 静态代码块在非静态代码块之前执行(静态代码块—>非静态代码块—>构造方法)。 该类不管创建多少对象，静态代码块只执行一次.
3. **静态内部类（static修饰类的话只能修饰内部类）：** 静态内部类与非静态内部类之间存在一个最大的区别: 非静态内部类在编译完成之后会隐含地保存着一个引用，该引用是指向创建它的外围类，但是静态内部类却没有。没有这个引用就意味着：1. 它的创建是不需要依赖外围类的创建。2. 它不能使用任何外围类的非static成员变量和方法。
4. **静态导包(用来导入类中的静态资源，1.5之后的新特性):** 格式为：import static 这两个关键字连用可以指定导入某个类中的指定静态资源，并且不需要使用类名调用类中静态成员，可以直接使用类中静态成员变量和成员方法。

**this 关键字**

this关键字用于引用类的当前实例。 例如：



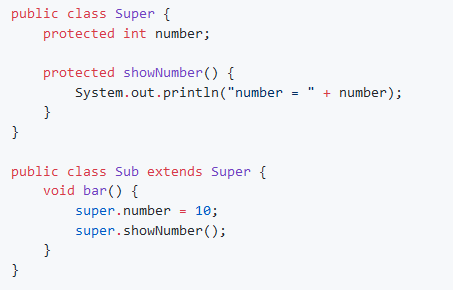
在上面的示例中，this关键字用于两个地方：

* this.employees.length：访问类Manager的当前实例的变量。
* this.report（）：调用类Manager的当前实例的方法。

此关键字是可选的，这意味着如果上面的示例在不使用此关键字的情况下表现相同。 但是，使用此关键字可能会使代码更易读或易懂。

**super 关键字**

super关键字用于从子类访问父类的变量和方法。 例如：



在上面的例子中，Sub 类访问父类成员变量 number 并调用其其父类 Super 的 showNumber（） 方法。

**使用 this 和 super 要注意的问题：**

* super 调用父类中的其他构造方法时，调用时要放在构造方法的首行！this 调用本类中的其他构造方法时，也要放在首行。
* this、super不能用在static方法中。

**简单解释一下：**

被 static 修饰的成员属于类，不属于单个这个类的某个对象，被类中所有对象共享。而 this 代表对本类对象的引用，指向本类对象；而 super 代表对父类对象的引用，指向父类对象；所以， **this和super是属于对象范畴的东西，而静态方法是属于类范畴的东西**。

35、