Java basic questions

# the different of static between final?

* 1. static关键字

static 关键字可以用来修饰类的变量，方法和内部类。static 是静态的意思，也是全局的意思它定义的东西，属于全局与类相关，不与具体实例相关。就是说它调用的时候，只是 ClassName.method(),而不是 new ClassName().method()。new ClassName()不就是一个对象了吗？static 的变量和方法不可以这样调用的。它不与具体的实例有关。

* 1. final关键字

final 关键字有三个东西可以修饰的。修饰类，方法，变量。  详细解释一下：    
在类的声明中使用 final    
使用了 final 的类不能再派生子类，就是说不可以被继承了。有些 java 的面试题里面，问 String 可不可以被继承。答案是不可以，因为 java.lang.String是一个 final 类。这可以保证 String 对象方法的调用确实运行的是 String 类的方法，而不是经其子类重写后的 方法。    
在方法声明中使用 final    
被定义为 final 的方法不能被重写了，如果定义类为 final 的话，是所有的方法都不能重写。而我们只需要类中的某几个方法，不可以被重写，就在方法前加 final 了。而且定义为 final 的方法执行效率要高的啊。    
在变量声明中使用 final    
这样的变量就是常量了，在程序中这样的变量不可以被修改的。修改的话编译器会抱错的。而且执行效率也是比普通的变量要高。final 的变量如果没有赋予初值的话，其他方法就必需给他赋值，但只能赋值一次。

注意：子类不能重写父类的静态方法哦，也不能把父类不是静态的重写成静态的方法。想隐藏父类的静态方法的话，在子类中声明和父类相同的方法就行了。

# String,StringBuffer,StringBuilder?

***1****.*三者在**执行速度**方面的比较：**StringBuilder >  StringBuffer  >  String**

***2****.***String <（StringBuffer，StringBuilder）**的原因

　　　　String：字符串常量

　　　　StringBuffer：字符创变量

　　　　StringBuilder：字符创变量

　　从上面的名字可以看到，String是“字符创常量”，也就是不可改变的对象。对于这句话的理解你可能会产生这样一个疑问  ，比如这段代码：

1 String s = "abcd";  
2 s = s+1;  
3 System.out.print(s);// result : abcd1

 　　我们明明就是改变了String型的变量s的，为什么说是没有改变呢? 其实这是一种欺骗，JVM是这样解析这段代码的：首先创建对象s，赋予一个abcd，然后再创建一个新的对象s用来执行第二行代码，也就是说我们之前对象s并没有变化，所以我们说String类型是不可改变的对象了，由于这种机制，每当用String操作字符串时，实际上是在不断的创建新的对象，而原来的对象就会变为垃圾被ＧＣ回收掉，可想而知这样执行效率会有多底。

　　　  而StringBuffer与StringBuilder就不一样了，他们是字符串变量，是可改变的对象，每当我们用它们对字符串做操作时，实际上是在一个对象上操作的，这样就不会像String一样创建一些而外的对象进行操作了，当然速度就快了。

***3****.*一个特殊的例子：

1 String str = “This is only a” + “ simple” + “ test”;  
3 StringBuffer builder = new StringBuilder(“This is only a”).append(“ simple”).append(“ test”);

 你会很惊讶的发现，生成str对象的速度简直太快了，而这个时候StringBuffer居然速度上根本一点都不占优势。其实这是JVM的一个把戏，实际上：

　　String str = “This is only a” + “ simple” + “test”;

　　其实就是：

　　String str = “This is only a simple test”;

　　所以不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的String对象的话，速度就没那么快了，譬如：

　　String str2 = “This is only a”;

　　String str3 = “ simple”;

　　String str4 = “ test”;

　　String str1 = str2 +str3 + str4;

　　这时候JVM会规规矩矩的按照原来的方式去做。

***4.*StringBuilder与 StringBuffer**

**StringBuilder：线程非安全的**

**StringBuffer：线程安全的**

　　当我们在字符串缓冲去被多个线程使用是，JVM不能保证StringBuilder的操作是安全的，虽然他的速度最快，但是可以保证StringBuffer是可以正确操作的。当然大多数情况下就是我们是在单线程下进行的操作，所以大多数情况下是建议用StringBuilder而不用StringBuffer的，就是速度的原因。

**对于三者使用的总结**：

1.如果要操作少量的数据用 = 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**

# == and equal?

int a =3 , b = 3;  
Integer i = new Integer(3);  
Integer m = new Integer(3);  
System.out.println(i==m);//false（1）  
System.out.println(a==i);//true（2）  
  
//int a = b =3 ;//在java中会编译报错  
System.out.println(a==b);//true（3）  
String s1 ="Monday";  
String s2 ="Monday";  
System.out.println(s1==s2);//true（4）  
System.out.println(s1.equals(s2));//true（5）  
  
String s11 = "Monday";  
String s22 = new String("Monday");  
System.out.println(s11==s22);//false（6）  
System.out.println(s11.equals(s22));//true（7）

1.（1）说明new方法在线程区变量中分配内存。与（2）比较，

int a = 3;  
int b = 3；  
编译器先处理int a = 3；首先它会在栈中创建一个变量为a的引用，然后查找栈中是否有3这个值，如果没找到，就将3存放进来，然后将a指向3。接着处理int b = 3；在创建完b的引用变量后，因为在栈中已经有3这个值，便将b直接指向3。这样，就出现了a与b同时均指向3的情况。这时，如果再令a=4；那么编译器 会重新搜索栈中是否有4值，如果没有，则将4存放进来，并令a指向4；如果已经有了，则直接将a指向这个地址。因此a值的改变不会影响到b的值。要注意这 种数据的共享与两个对象的引用同时指向一个对象的这种共享是不同的，因为这种情况a的修改并不会影响到b, 它是由编译器完成的，它有利于节省空间。而一个对象引用变量修改了这个对象的内部状态，会影响到另一个对象引用变量。  
String是一 个特殊的包装类数据。可以用：  
String str = new String("abc");  
String str = "abc";  
两种的形式来创建，第一种是用new()来新 建对象的，它会在存放于堆中。每调用一次就会创建一个新的对象。

－＞ **String str = new String("abc");自己补充： 应该说有会产生两个对象，一个为new String("abc")的实体对象放到内存堆中， 一个为堆栈对象 ｓｔｒ 也就是类实例对象的引用对象。**

而第二种**（**String str = "abc";**）**是先在栈中创建一个对String类的对象引用变量str，然后查找栈 中有没有存放"abc"，如果没有，则将"abc"存放进栈，并令str指向”abc”，如果已经有”abc” 则直接令str指向“abc”。  
**比较类里面的数值是否相等时，用equals()方法**；当 **测试两个包装类的引用是否指向同一个对象时，用==，** 下面用例子说明上面的理论。  
String str1 = "abc";  
String str2 = "abc";  
System.out.println(str1==str2); //true  
**可以看出str1和 str2是指向同一个对象的。**String str1 =new String ("abc");  
String str2 =new String ("abc");  
System.out.println(str1==str2); // false  
**用new的方式是生成不同的对象。每一次生成一个** 。  
   因此用第二种方式（String str = "abc";）创建多个”abc”字符串,在内存中其实只存在一个对象而已. 这种写法有利与节省内存空间. 同时它可以在一定程度上提高程序的运行速度，因为JVM会自动根据栈中数据的实际情况来决定是否有必要创建新对象。而对于String str = new String("abc")；的代码，则一概在堆中创建新对象，而不管其字符串值是否相等，是否有必要创建新对象，从而加重了程序的负担。  
   另一方面, 要注意: 我们在使用诸如String str = "abc"；的格式定义类时，总是想当然地认为，创建了String类的对象str。担心陷阱！对象可能并没有被创建！而可能只是指向一个先前已经创建的 对象。只有通过new()方法才能保证每次都创建一个新的对象。由于String类的immutable性质，当String变量需要经常变换其值时，应 该考虑使用StringBuffer类，以提高程序效率。

# final、finally和finalize的区别？

# **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。**

# **finally 是异常处理语句结构的一部分，表示总是执行。**

# **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**

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。  
  finally—在异常处理时提供 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）。  
  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。