面试题

面试题：

Short s = 1 ;

s = s + 1; s += 1;两个表达式是否有问题？

s = s + 1 ;//有问题，因为short在参与运算的时候会转换成int

然后无法赋值给s。

Short s = 1 ;

s += 1 ;//无报错，因为扩展的赋值运算符隐含了 强制类型转换

int a = 10 ;

int b = 20;

int c = ( a = b );//把b赋值给a然后留下a的值赋值给c

面试题：

请自己实现两个整数变量的交换

方式1：(开发用这个)

int a = 10 ;

int b = 20 ;

int t ;

t = a ;

a = b ;

b = t ;

方式2：用位异或实现（面试用）

左边：a,b,a

右边：a ^ b

a = a ^ b ;//将此式带入下面

b = a ^ b ;//a^b^b = a

a = a ^ b ;//a^b^a = b

方式3：用变量相加的方法

a = a + b;

b = a – b ;

a = a – b ;

方式4：一句话搞定

b = (a+b) – (a=b);

面试题：

请用最有效率的方式写出计算2乘以8的结果

2<<3

面试题：

获取两个整数中的最大值

Int x = 100 ;

Int y = 200 ;

Int max = (x>y)?x:y;

获取三个整数中的最大值

Int x = 100 ;

Int y = 200 ;

Int z = 300 ;

方式1：int max = ((x>y)?x:y)>z?((x>y)?x:y):z;

方式2：int max = (a>b)?((a>c)?a:c):((b>c)?b:c);

测试程序的时候，要考虑三个数据：正确数据，错误数据，边界数据

面试题：

byte可以作为switch的表达式吗？可以

long可以作为switch的表达式吗？不可以

String可以作为switch的表达式吗？JDK7以后可以

int x=2;

int y=3;

switch(x){

default:

y++;

case 3:

y++;

case 4:

y++

}

System.out.println(y);//y=6

先找两个case都不符合，所以y值不变进入default，在default处y++，由于没有遇到break所以程序继续，后面还有两个y++

在控制台输出所有的水仙花数，并统计其个数

输出乘法口诀表

求原码为10110100B的原反补（B结尾说明是2进制）

原码:10110100

反码:11001011

补码:11001100

求补码为11101110B的原码

补码:11101110

反码:11101101

原码:10010010

练习：

byte b = 130 ;//报错，超出范围

byte b = (byte)130 ;//可以执行，但是b的结果是-126

过程：130是int类型,在内存中储存为

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

强制截取byte就是8位，变成了

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

于是这个数字10000010就是所得数字的补码求反码和原码

反码:10000001

原码:11111110=-126

所以值为-126

练习:byte b = (byte)300;

300=100101100

截取8位00101100，正数原反补是他本身所以是44

练习:short s = (short)70000;

70000=10001000101110000

截取16位是0001000101110000，正数原反补是他本身4464

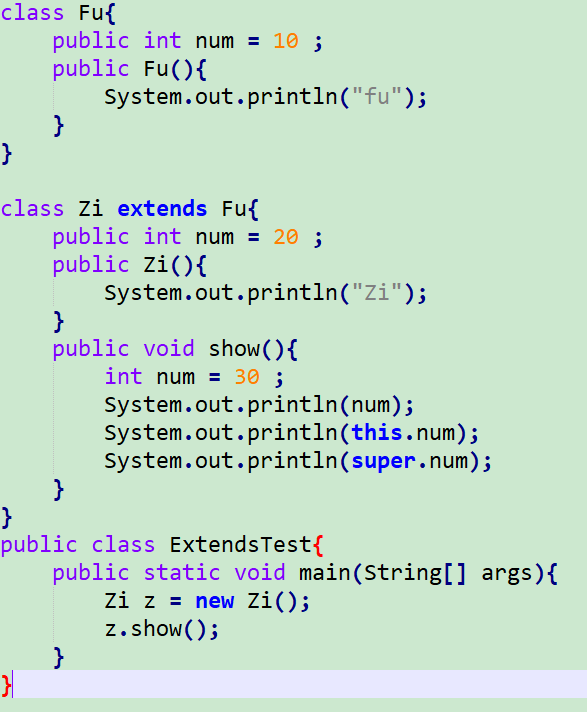
静态代码块，构造代码块，构造方法的执行顺序

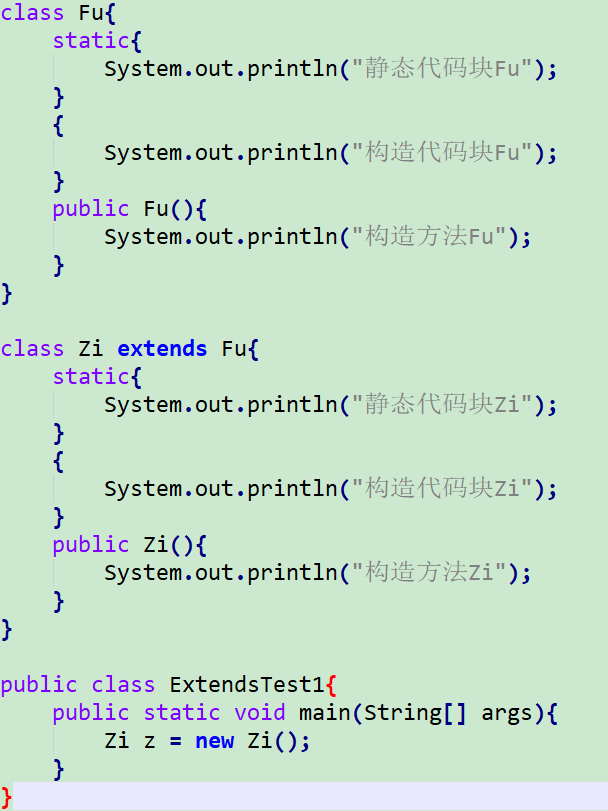
静态代码块 -- 构造代码块 – 构造方法

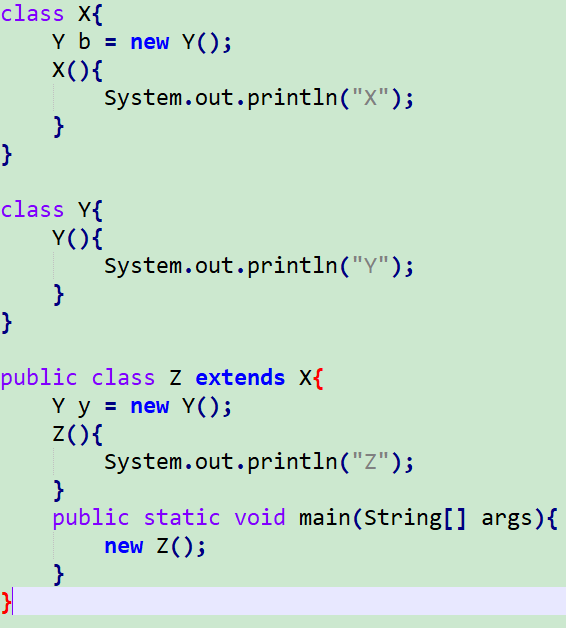
静态代码块：只执行一次

构造代码块：每次调用构造方法都执行

看程序写结果







方法重载和重写有什么区别？方法重载能改不返回值类型吗？

重写Override:在子类中出现和父类中一模一样的方法声明的现象

重载Overload:同一个类中出现的方法名相同参数列表不同的现象

方法重载能改变返回值类型，因为它和返回值类型无关

this关键字和super关键字的区别，以及各自的场景和使用

this:代表当前类的对象引用

super:代表父类存储空间的标识（可以理解为父类的引用，通过这个东西可以访问父类的成员）

场景：成员变量 构造方法 成员方法

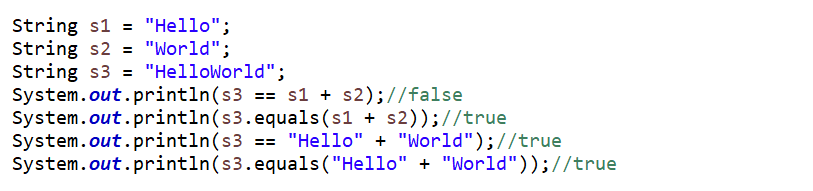
final修饰局部变量的问题

final可以修饰局部变量，修饰基本数据类型时，基本类型的值不可改变。修饰引用数据类型时，引用数据类型的地址值不能改变

final修饰变量的时机

在构造方法完毕前赋值

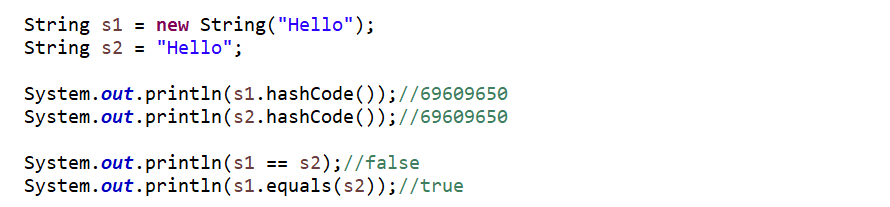
面试题：看程序写结果



第一条语句，因为s1+s2是两个变量相加，会先开辟空间，所以新开辟的空间地址值与s3的地址值不同

第二条第四条略

第三条因为”Hello”和”World”是常量，所以会先相加再比较，相加之后是”HelloWorld”，所以会在字符串常量池中寻找，找到了，所以与s3指向的是同一个字符串，所以地址值相同



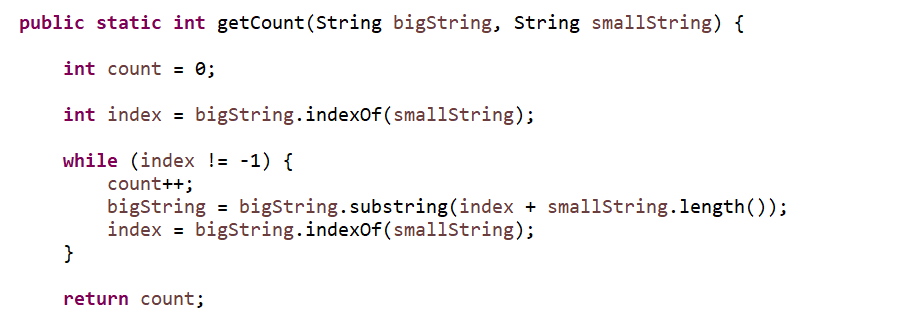
解释：第一行在堆内存中生成String对象，对象储存的值为”Hello”，此时new出来的String对象会去字符串常量池寻找”Hello”，如果有，则会把该字符串的地址值存在堆内存的new String()中，如果没有，则会在字符串常量池创建”Hello”然后把地址值给new String()并且此时s1指向的是堆内存中new String()对象的地址值，接下来创建s2字符串，直接赋值”Hello”，此时会去字符串常量池找”Hello”，之前有创建，所以直接让s2指向该地址值；虽然s1和s2的地址值不一样，但是String类重写了hashCode方法，String类的hashCode方法是返回字符串的哈希码值，所以此时s1和s2的hashCode是一样的，但是s1==s2比较地址值，所以是false，而s1.equals(s2)比较字符串内容是true



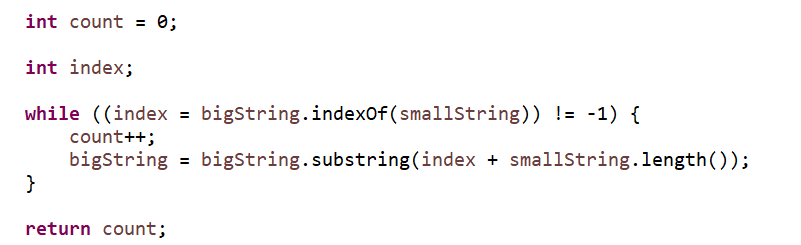
区别：如果独立运行前者会创建两个对象，后者会创建一个对象

练习题：统计大串中小串出现的次数

思路：首先查找一下大串中是否有小串，没有，则直接返回0次，如果有那就从这个字符串开始截取，然后查截取后的字符串中是否有小串直到没有为止



代码优化：



StringBuffer类面试题

String,StringBuffer,StringBuilder区别

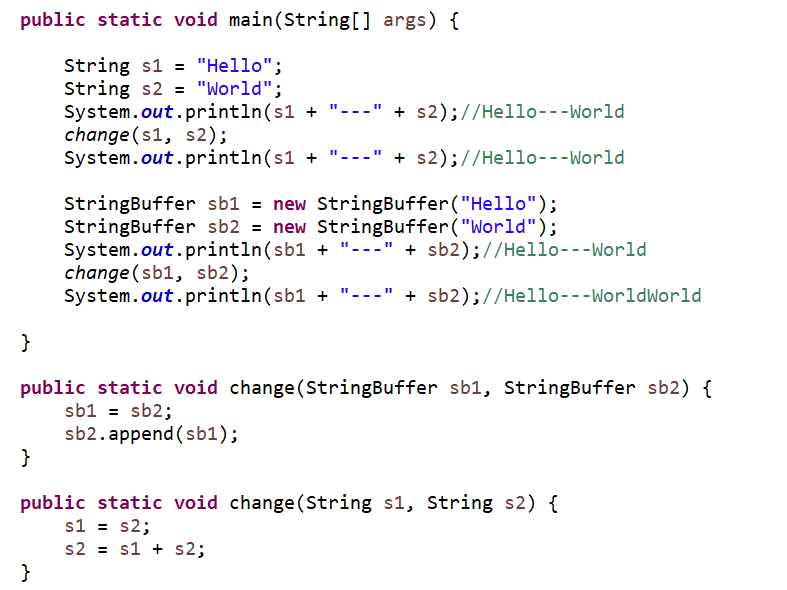
A:String是内容不可变的，而StringBuffer,StringBuilder都是内容可变的

B:StringBuffer是同步的，数据安全，效率低;StringBuilder是不同步的,数据不安全，效率高

StringBuffer和数组的区别

二者都可以看成是一个容器，装其他的数据；但是StringBuffer数据最终是一个字符串数据，而数组可以放置多种数据，但必须是同一种数据类型

看程序写结果



解释：第一个和第三个输出语句不做解释；

第一个change(s1,s2);方法调用，虽然传递的是String类型是引用类型，但是String比较特殊，他是常量值，所以String是特殊的引用类型，可以当做基本类型看，形参的改变对实参没有影响，形参复制的是字符串的值，并不是拿地址操作

第二个change(sb1.sb2);方法调用，当StringBuffer作为形参传递的时候，传递的是地址值，第一个s1=s2;是把s2的地址给了s1，然后此时s1也是World，所以s2.append(s1);然后s2就会返回worldworld；但是s1并没有改变，所以主方法中会显示这样的结果;调用方法，是可以改变形参的，所以建议调用方法来改变

注意：String作为形参传递效果和基本类型作为形参传递是一样的

把String转换成int类型

String s = “100”;

int a = Integer.parseInt(s);

把int类型转换成String类型

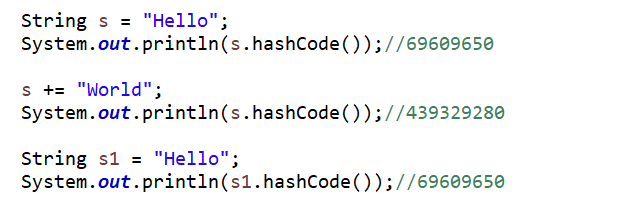
int a = 100;

String s = String.valueOf(a);

八个基本类型对应的类都有parseInt方法，而且八个基本类型对应的类也都有valueOf方法，不用valueOf方法的原因是因为该方法返回的是一个Integer类型，而不是int类型如下：

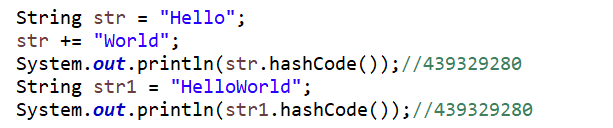
**public static int parseInt​(**[**String**](mk:@MSITStore:C:\Users\ylx05\Desktop\jdk-9_汉化版.CHM::/java/lang/String.html)**s)**

**public static**[**Integer**](mk:@MSITStore:C:\Users\ylx05\Desktop\jdk-9_汉化版.CHM::/java/lang/Integer.html)**valueOf​(**[**String**](mk:@MSITStore:C:\Users\ylx05\Desktop\jdk-9_汉化版.CHM::/java/lang/String.html)**s)**

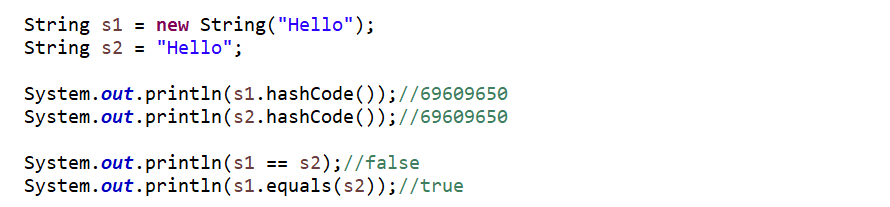


字符串直接赋值的方式是先到字符串常量池里去找，如果有就直接返回，没有，就创建并返回

一旦被赋值就不能改变



理解：新建str字符串，会首先去字符串常量池寻找是否有Hello，如果有，则str直接指向该字符串地址，如果没有，则创建Hello字符串(Hello字符串一旦被创建，则无法改变)然后又会去创建World字符串，接着把Hello和World拼接起来，把拼接起来的完整字符串地址给str，自始至终Hello字符串的值未曾改变，str指向的地址值可以变，当再创建一个str1并赋值HelloWorld时候，会找到之前已经存在的HelloWorld，而不会再创建，所以两个字符串输出的hashCode相同

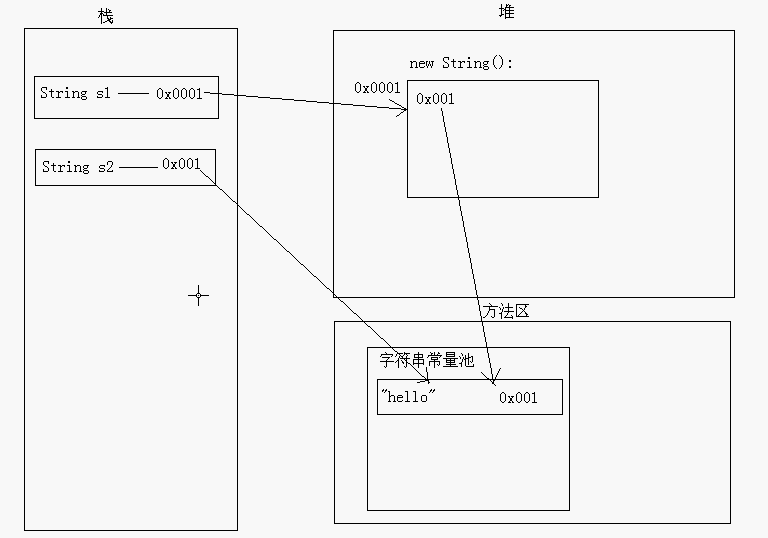


解释：第一行在堆内存中生成String对象，对象储存的值为”Hello”，此时new出来的String对象会去字符串常量池寻找”Hello”，如果有，则会把该字符串的地址值存在堆内存的new String()中，如果没有，则会在字符串常量池创建”Hello”然后把地址值给new String()并且此时s1指向的是堆内存中new String()对象的地址值，接下来创建s2字符串，直接赋值”Hello”，此时会去字符串常量池找”Hello”，之前有创建，所以直接让s2指向该地址值；虽然s1和s2的地址值不一样，但是String类重写了hashCode方法，String类的hashCode方法是返回字符串的哈希码值，所以此时s1和s2的hashCode是一样的，但是s1==s2比较地址值，所以是false，而s1.equals(s2)比较字符串内容是true

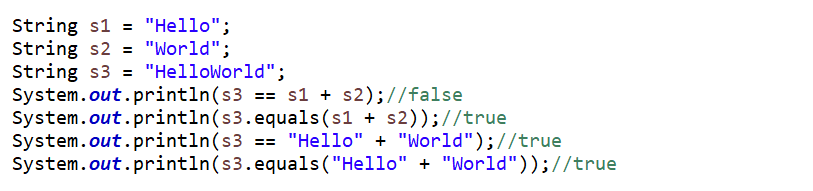


区别：如果独立运行前者会创建两个对象，后者会创建一个对象

内存模型图如下



面试题：看程序写结果

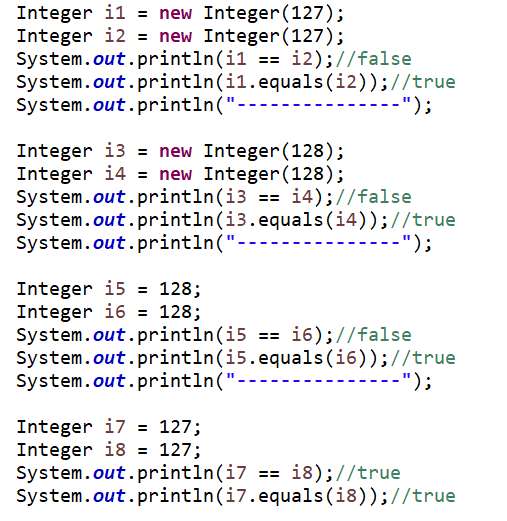


第一条语句，因为s1+s2是两个变量相加，编译的时候不知道变量的值，会先开辟空间，所以新开辟的空间地址值与s3的地址值不同

第二条第四条略

第三条因为”Hello”和”World”是常量，所以会先相加再比较，相加之后是”HelloWorld”，所以会在字符串常量池中寻找，找到了，所以与s3指向的是同一个字符串，所以地址值相同

面试题：

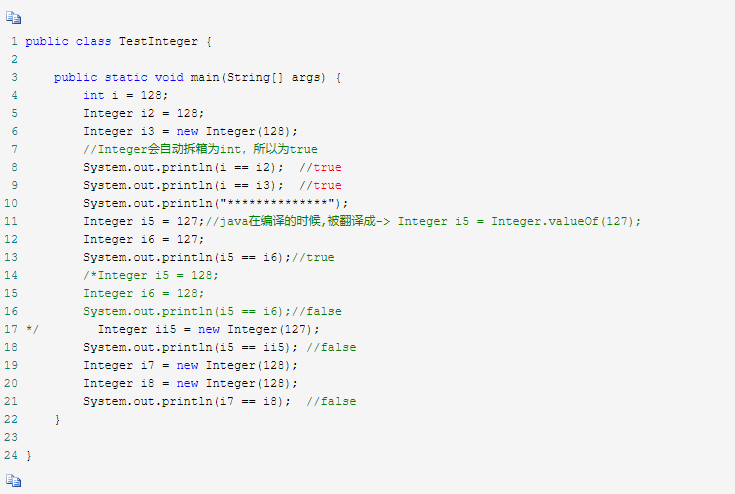


解释：第一句false是因为i1和i2都是指向不同的对象，地址值不同；第三句同理；第五句，因为存在自动装箱操作，在底层运行的时候是Integer i5 = Integer.valueOf(128);实则也创建了对象，所以i5==i6也是false；第七句，查看原码得知，-128到127之间有一个byte缓冲区，如果数据范围在这个范围，并不会创建新的对象

注意：Integer的数据直接赋值，如果在-128到127之间，会直接从缓冲池获取数据

  Integer与int的区别我们耳熟详的有两点：1.Integer是int的包装类。2.Integer的默认初始值是null，而int的默认初试值是0。

     下面通过代码进行详细比较。



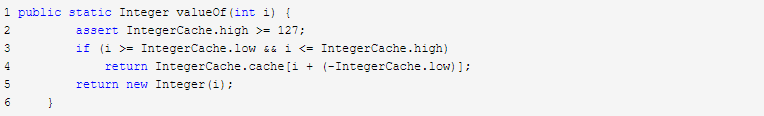
    第8行和第9行的结果都是true，因为Integer会拆箱为int（jdk 1.5以上）。

什么时候会拆箱？——基本数据类型和引用数据类型做运算时

什么时候会装箱？——基本数据类型赋值给引用数据类型的时候

    第13行的结果为true，而第16行的结果为false。很奇怪~因为java在编译Integer i5=127的时候，被翻译成Integer i5=Integer.valueOf(127);

    所以关键就是看valueOf()函数了。JDK源码的valueOf()是这样的：



    valueOf：public static Integer valueOf(int i) 返回一个表示指定的int值的Integer实例。如果不需要新的Integer实例，则通常应优先使用该方法，而不是构造方法Integer(int),因为该方法有可能通过缓存经常请求的值而显著提高空间和时间性能。

    对于-128到127之间的数，会进行缓存，Integer i5=127时，会将127进行缓存，下次再写Integer i6=127时，就直接从缓存中取，就不会new了。所以第13行的结果为true，而第16行的结果为false。

    对于第18和21行，对象不一样，都为false。

    综上所述：

    ①无论如何，Integer和new Integer不会相等。不会经历拆箱过程，前者指向常量池，后者的引用指向堆。他们的内存地址不一样。

    ②两个都是非new出来的Integer，如果数在-128到127之间，则是true，否则为false。

    ③两个都是new出来的，为false。

    ④int和Integer（无论new 否），都为true，因为会把Integer自动拆箱为int再比较。

写一个程序，获取任意区间的随机数

return (int)(Math.random() \* (max – min + 1) + min);

解释：如，取20-50之间的随机数，50-20+1=31，random()出来的数乘以31然后(int)一下，范围就是0-30的整数，然后加20正好就是20-50之间的数

写一个工具类，把日期和字符串相互转换；

把字符串转换成日期；把日期转换成字符串；

制作日期工具类，计算你来世界多少天

简便方法算出任意年份的2月有多少天



"aababcabcdabcde",获取字符串中每一个字母出现的次数要求结果:a(5)b(4)c(3)d(2)e(1)

思路：

A:定义一个字符串

B:定义一个Map<Character,Integer>集合

C:把字符串转化成字符数组

D:遍历字符数组，得到每一个字符

E:拿这个字符到Map集合中去找，看返回值

如果返回null,则把该字符作为键，1作为值存储

不是返回Null,然后重新存储该键，值加1

F:遍历TreeMap集合

HashMap和Hashtable的区别

Hashtable:线程安全，效率低；不允许null键和null值

HashMap:线程不安全，效率高；允许null键和null值

List,Set,Map等接口是否都继承与Map接口

List,Set不是继承自Map的接口，它们继承自Collection接口

Map接口本身就是一个顶层接口

Collection和Collections的区别

Collection:是单列集合的顶层接口，有字接口List和Set

Collections:是针对集合操作的工具类，有对集合进行排序和二分查找的方法

面试题：

final，finally和finalize的区别

final：最终的意思，可以修饰类，成员变量，成员方法

修饰类，类不能被继承

修饰变量，变量是常量

修饰方法，方法不能被重写

finally：是异常处理的一部分，用于释放资源，一般来说，代码肯定会执行，特殊情况：在执行到finally之前jvm就退出了

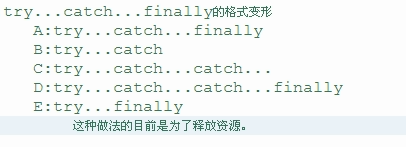
finalize：是Object类的一个方法，用于垃圾回收

如果catch里面有return语句，请问finally里面的代码还会执行吗？如果会，是在return之前还是在return之后

答：会执行，在return之前执行



此程序返回的是30



run()和start()的区别：

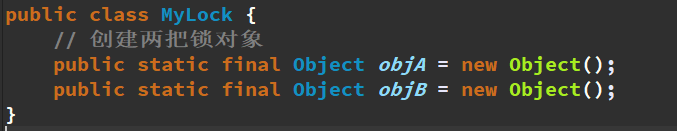
run()仅仅是封装被线程执行的代码，直接调用就是普通方法

start()首先启动了线程，然后由jvm去调用该线程的run()方法

面试题

写一段死锁代码

MyLock.java



DieLock.java



DieLockDemo.java

