## Java 局部内部类使用局部final变量

1. 内部类是外部类的一个成员，就像外部类的成员方法一样，所以内部类有权限访问外部类的所有成员，包括private的。
2. 内部类不能访问外部类方法中的局部变量，除非变量是final的(一般发生在方法中定义的内部类)。这是因为局部变量的生命周期原因。
3. 所谓“局部内部类”就是在对象的方法成员内部定义的类。而方法中的类，访问同一个方法中的局部变量，却必须要加上一个final。

原因是编译程序实现上的困难：

* 内部类对象的生命周期会超过局部变量的生命期。局部变量的生命期：当该方法被调用时，该方法中的局部变量在栈中被创建，当方法调用结束时，退栈，这些局部变量全部死亡。而内部类对象生命期，与其它类一样，当创建一个局部内部类对象后，只有当没有其它人再引用它时，它才能死亡。所以完全可能一个方法已调用结束（局部变量已死亡），但该局部类的对象仍然活着。即：局部类的对象生命期会超过局部变量。
* 局部内部类的对象访问同一个方法中的局部变量，那么这就要求只要局部内部类对象还活着，那么栈中的那些它要访问的局部变量就不能“死亡”（否则：它都死了，还访问个什么呢？）。这就是说：**局部变量的生命期至少等于或大于局部内部类对象的生命期。**
* 解决方法：局部内部类的对象可以访问同一个方法中被定义为final的局部变量。定义为final后，编译程序的实现方法：将所有的局部内部类对象要访问的final型局部变量，都**拷贝**成为该内部类对象中的一个数据成员。这样，即使栈中局部变量（含final）已死亡，但由于它是final,其值永不变，因而局部内部类对象在变量死亡后，照样可以访问final型局部变量
* 归纳总结：局部内部类对象中包含有要访问的final型局部变量的一个**拷贝**，成为它的数据成员。因此，正是在这个意义上，final型局部变量的生命期，超过其方法的一次调用。**严格来说，方法调用结束，所有的局部变量（含final）全死亡了。但：局部内部类对象中有final型局部变量的拷贝。**

## java 24 \* 60 \* 60 \* 1000（一天）

Date d=new Date();

SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd");

System.out.println("today:"+df.format(d));

System.out.println("the day **befor yestoday**:" + df.format(new Date(**d.getTime() - (long)2 \* 24 \* 60 \* 60 \* 1000)**));

System.out.println("the day **after tomorrow**:" + df.format(new Date(**d.getTime() + (long)3 \* 24 \* 60 \* 60 \* 1000)**));

## Java中子类和父类的初始化顺序-静态优先，父类优先

类的初始化顺序是：

1、初始化父类中的静态成员变量和静态代码块。

2、初始化子类中的静态成员变量和静态代码块。

3、初始化父类中的普通成员变量和代码块，在执行父类中的构造方法。

4、初始化子类中的普通成员变量和代码块，在执行子类中的构造方法。

## Java transient关键字

java语言的关键字，变量修饰符，如果用transient声明一个实例变量，当对象存储时，它的值不需要维持。换句话来说就是，用transient关键字标记的成员变量不参与序列化过程。

## Java >> n & << n

>> n （右移 除2的n次方）

<< n （左移 乘2的n次方）

## Java Collection接口的toArry方法注意

1. public Object[] toArray()方法：方法中会重新构造一个Object[]数组，然后将集合中每个元素赋给数组，由于new的是Object数组，所以无法向上转型回之前的数组。
2. public <T> T[] toArray(T[] a)方法：返回传入的数组a指定的类型，a的length小于集合size时会利用反射重新构建一个指定类型数组，并把集合中所有元素赋给数组。