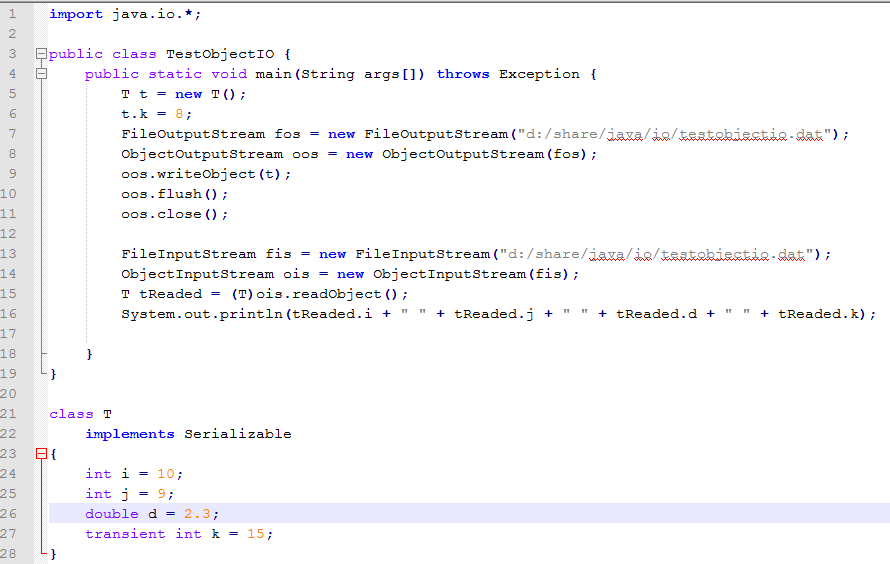
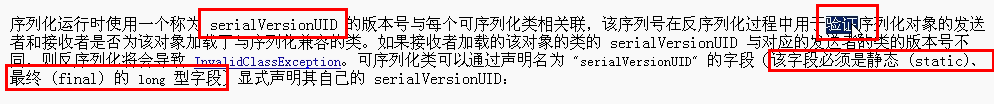
1. Object流：直接将Object写入或读出



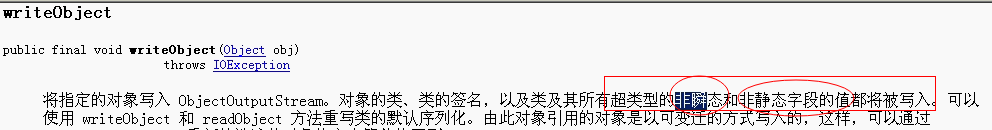
直接将类的对象转换(序列化)为字节流需要实现**Serializable**接口(标记性接口，没有定义方法)，读取顺序与写入顺序必须相同。

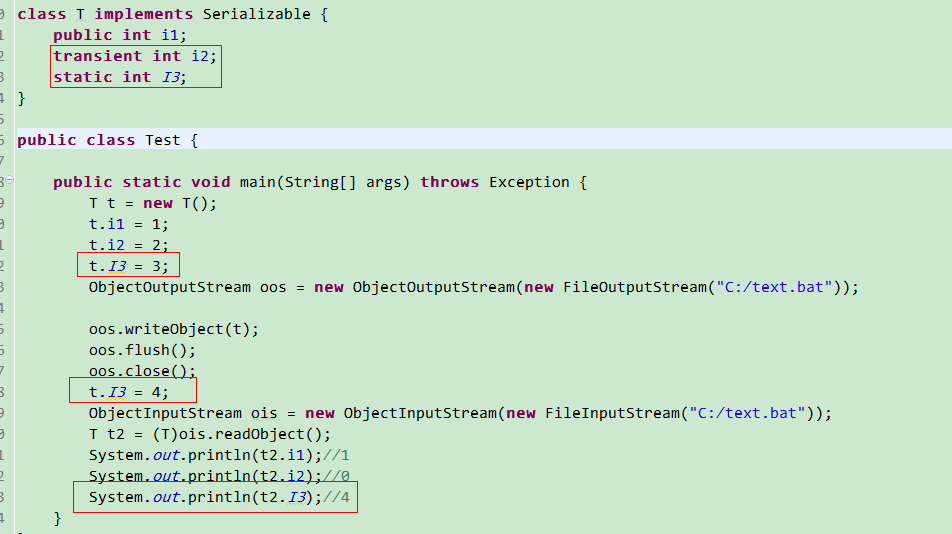
**Externalizable**接口：Serializable接口的子接口，其中含有readExternal(ObjectInput in)和writeEternal(ObjectOutput out)，可以自己控制对象的序列化过程。





1. 直接将类的对象转换(序列化)为字节流需要实现**Serializable**接口(标记性接口，没有定义方法)。Serializable接口又被成为marker或tag类的标记用接口，因为此接口没有任何方法需要实现。它唯一的目的就是声明有实现它的类是可以被序列化的。如果父类实现了Serializable接口，则子类也就自动的有实现，而不管是否有明确的声明。
2. 当对象被序列化时，被该对象引用的实例变量也会被序列化(该类也必须实现**Serializable接口**)，且所有被引用的对象(实现**Serializable接口**)也会被序列化，这些操作都是自动进行的。序列化操作是原子操作，也就是说：要么全有 ，要么全无。
3. **transient** 关键词修饰的成员变量在序列化的过程中不予考虑。解序列化时，该变量的值还原为系统默认的值(引用变量值为null、int为0、boolean为false等等)。注：静态变量不会被序列化，还原时，会维持类中原本的样子，而不是存储时的样子。

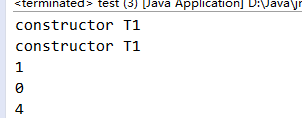




1. 解序列化的过程：
   1. 对象从stream流中读出来
   2. JVM通过存储的信息判断出对象的class类型
   3. JVM尝试寻找和加载对象的类，如果JVM找不到或无法加载该类，则JVM会抛出例外
   4. 新的对象会被配置在堆上，但构造函数不执行(这样会把对象的状态抹去又变成全新的)。
   5. 如果对象在继承树上有个不可序列化的祖先类，则该不可序列化类以及在它之上的类的构造函数(就算是可序列化也一样)就会执行。一旦构造函数连锁启动之后将无法停止。也就是说，从第一个不可序列化的父类开始，全部都会重新初始状态。
   6. 对象的实例变量会被还原成序列化时点的状态值。transient变量会被赋值为null的对象引用或原始数据类型的默认0、false等值。



输出：





输出：

