序列化：将对象转化为有序的字节序列，保存对象状态。

反序列化：将有序字节序列还原成对象，用于恢复对象状态。

反序列化必须有原类的模板才能还原。

序列化的目的：方便对象传输，持久化等。

**JDK实现序列化和反序列化**：

java.io.**ObjectOutputStream** : 对象输出流

writeObject(obj)

java.io.**ObjectInputStream** ：对象输入流

readObject(obj)

实现序列化的要求 ：

1. 实现 Serializable 接口 , 序列化接口没有方法和字段，仅标识可序列化的语义。实现此接口的子类均可序列化。若序列化的类含父类的字段，当想序列化改字段时，父类仍需序列化。**虽然没有什么字段和接口，但是序列化时会调用instanceof判断一个类是否实现类Seriablizable**
2. 实现 Externalizable ，Externalizable接口实现了Serializable , 含两个抽象方法: writeExtrenal(ObjectOutput) 和 readExternal(ObjectInput) 。如果没有实现这两个接口。**和Seriablizable接口的区别：E反序列化后字段内容为 空（初始值） 。 S反序列化后字段保留了序列化前的值。**

ObjectInput ： 扩展 DataInput 接口，用于从二进制字节流中读取字节

ObjectOuput : 扩展DataOutput接口，用于将java基本类型转换为一系列字节。

**Externalizable 接口反序列是调用了类的 无参构造器 创建一个对象后，再通过赋值填充到新对象，所以一定要有一个 无参构造器。**

**transient关键字**

阻止字段的序列化。值为初始值。

**序列化ID**

**两个类的序列化ID相同才能保证反序列的对象是一致private static final long serialVersionUID**

在序列化过程中，如果被序列化的类中定义了writeObject 和 readObject 方法，虚拟机会试图调用对象类里的 writeObject 和 readObject 方法，进行用户自定义的序列化和反序列化。

如果没有这样的方法，则默认调用是 ObjectOutputStream 的 defaultWriteObject 方法以及 ObjectInputStream 的 defaultReadObject 方法。

**关于ArrayList ：**

虽然，ArrayList底层实现的数组被标记为 transient ，但是仍可序列化。因为实现了writeObject() 和 readObject() ;这里面实现了自定的序列化机制。**与上面的理论一致。**

也就述说，用户自定义的 writeObject 和 readObject 方法可以允许用户控制序列化的过程，比如可以在序列化的过程中动态改变序列化的数值。我们发现ArrayList中有这两个方法的实现，那么基本可以确定，elementData能被序列化持久下来，肯定和这两个方法有关，虽然他被声明为transitent。

一句话 ： **ArrayList实现了writeObject 和 readObject，此而方法定义了elementData的序列化和反序列化策略。即便elementData用transient修饰。**

**why transient**

**why writeObject 和 readObject**