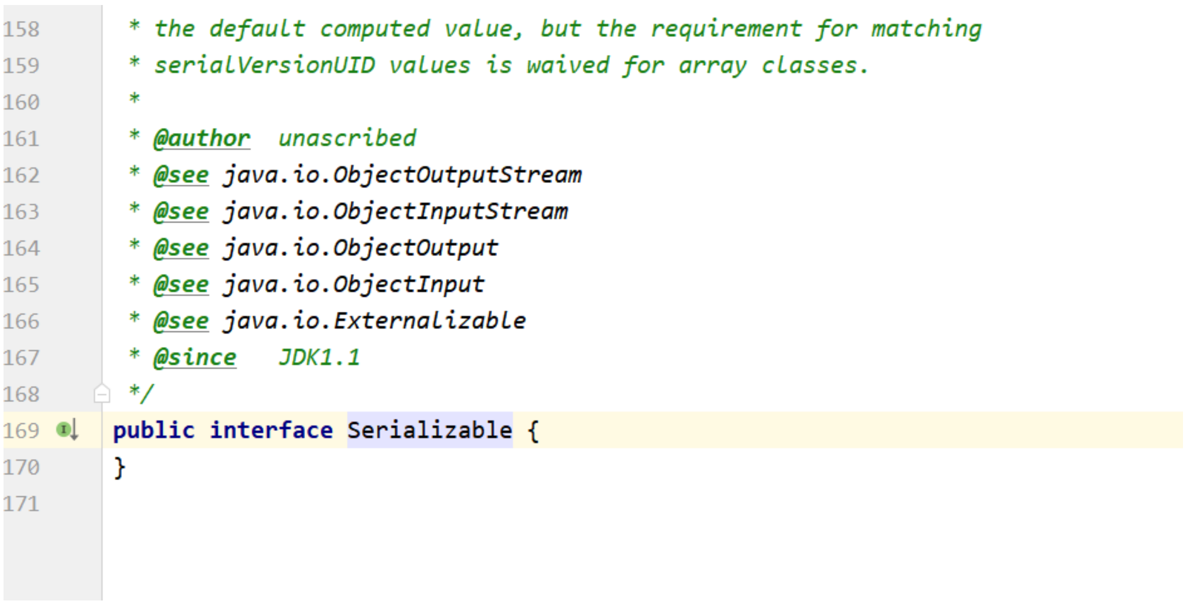
Serializable接口

1. 概念

java的序列化是 jdk1.1 时引入的特性，用于将 Java 对象转换为字节数组，便于存储或传输。并且，仍然可以将字节数组转换回 Java 对象原有的状态。

序列化的思想是 “冻结” 对象状态，然后写到磁盘或者在网络中传输；反序列化的思想是 “解冻” 对象状态，重新获得可用的 Java 对象



```java
 * the default computed value, but the requirement for matching
 * serialVersionUID values is waived for array classes.
 *
 * @author  unascribed
 * @see java.io.ObjectOutputStream
 * @see java.io.ObjectInputStream
 * @see java.io.ObjectOutput
 * @see java.io.ObjectInput
 * @see java.io.Externalizable
 * @since   JDK1.1
 */
public interface Serializable {
}
```

可以看到 **Serializable** 是一个空接口

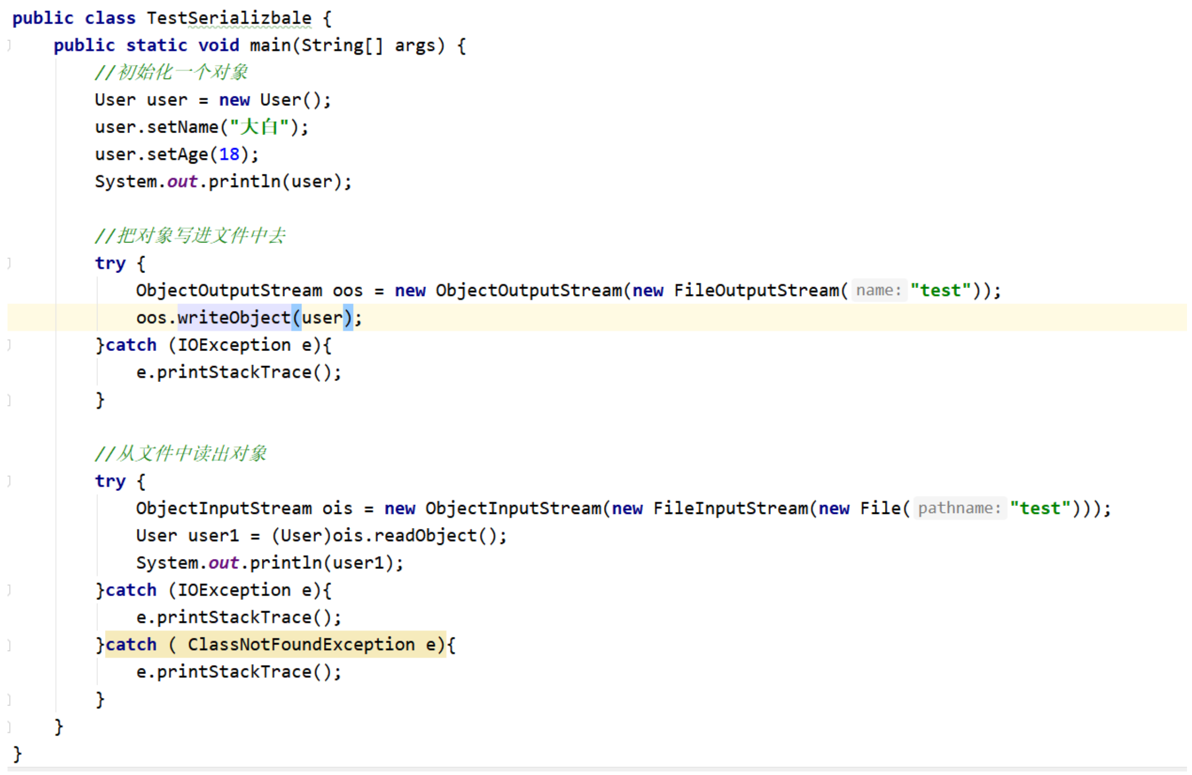
2.演示

创建一个 **User** 实体类

1569599870159

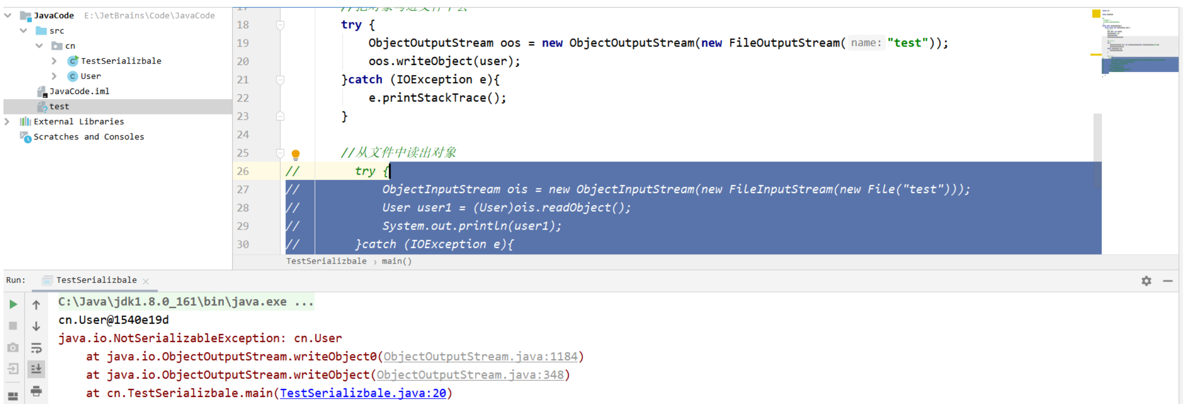
创建测试类，通过 ObjectOutputStream 将 user 对象写入到文件当中（序列化）； 再通过 ObjectInputStream 将 user 对象从文件中读取出来（反序列化）

测试类



```java
public class TestSerializbale {
    public static void main(String[] args) {
        //初始化一个对象
        User user = new User();
        user.setName("大白");
        user.setAge(18);
        System.out.println(user);

        //把对象写进文件中去
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User user1 = (User)ois.readObject();
            System.out.println(user1);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

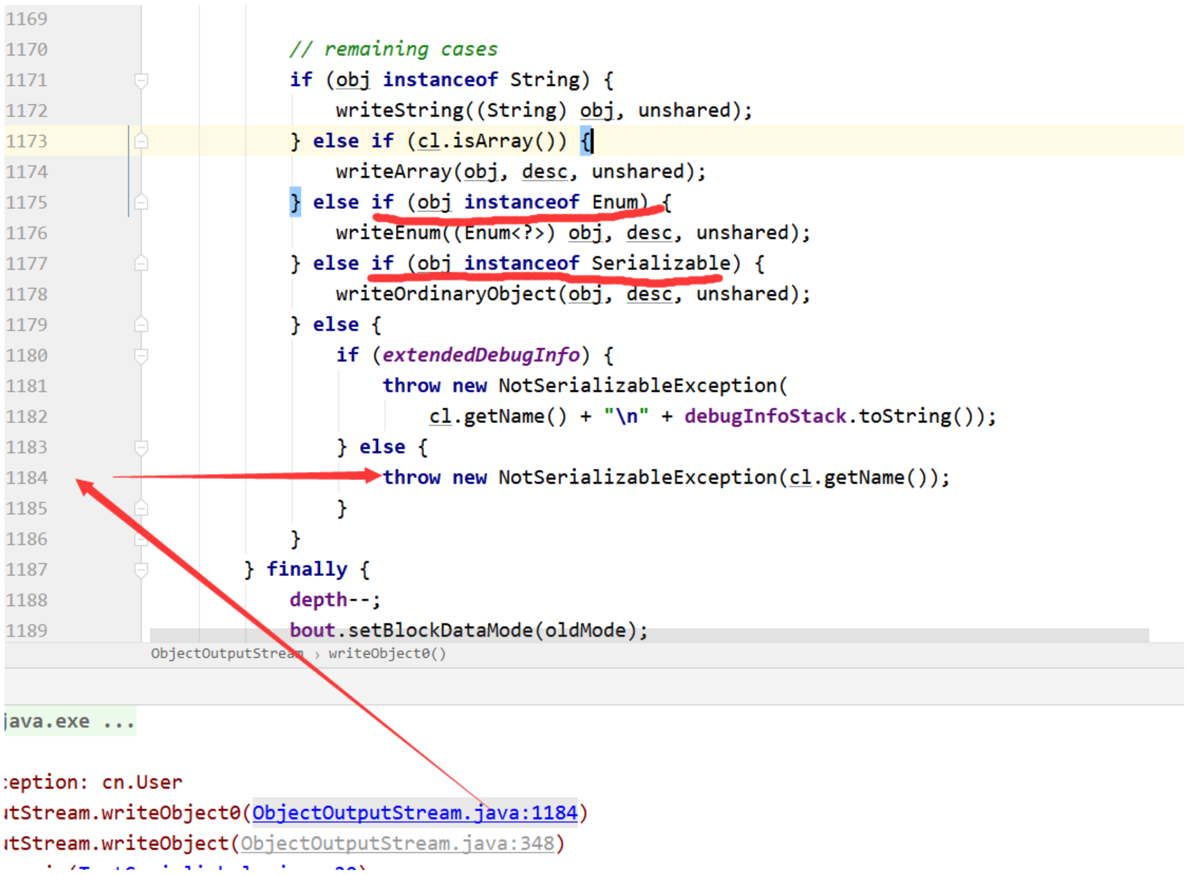
运行结果



```java
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
//        try {
//            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File("test")));
//            User user1 = (User)ois.readObject();
//            System.out.println(user1);
//        }catch (IOException e){
```

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
cn.User@1540e19d
java.io.NotSerializableException: cn.User
	at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1184)
	at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:348)
	at cn.TestSerializbale.main(TestSerializbale.java:20)
```

当我们点进报错信息里面



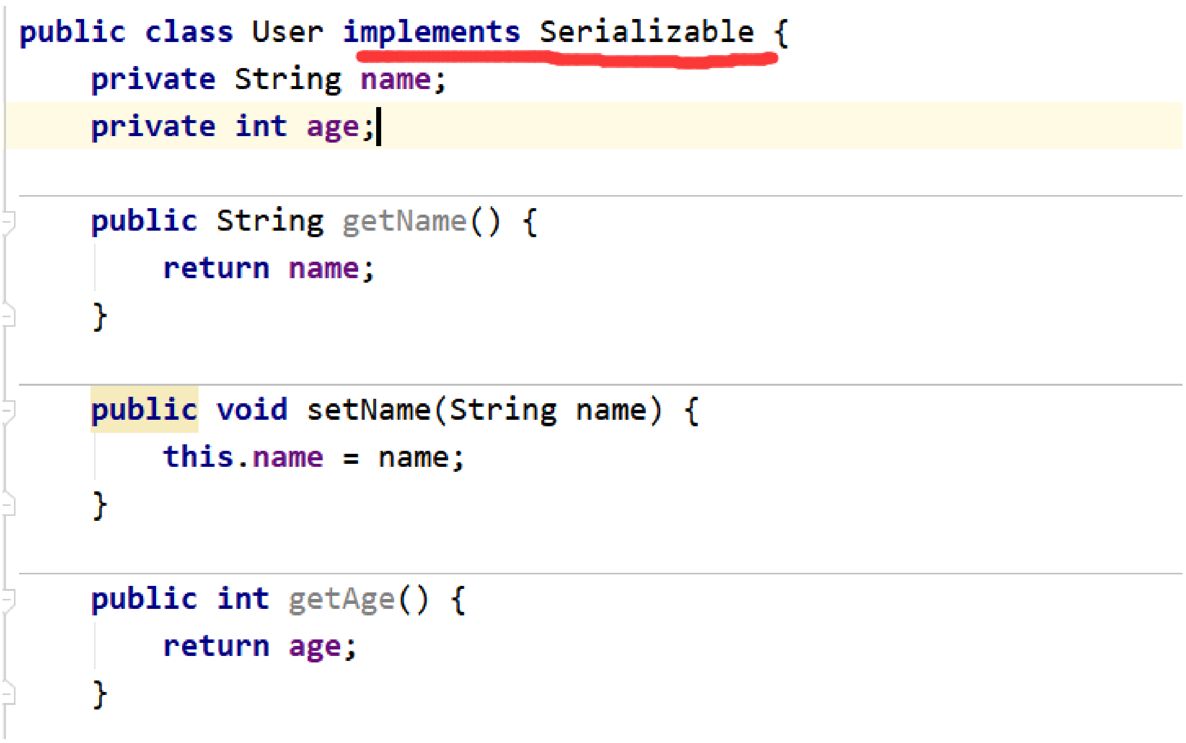
```
            // remaining cases
            if (obj instanceof String) {
                writeString((String) obj, unshared);
            } else if (cl.isArray()) {
                writeArray(obj, desc, unshared);
            } else if (obj instanceof Enum) {
                writeEnum((Enum<?>) obj, desc, unshared);
            } else if (obj instanceof Serializable) {
                writeOrdinaryObject(obj, desc, unshared);
            } else {
                if (extendedDebugInfo) {
                    throw new NotSerializableException(
                        cl.getName() + "\n" + debugInfoStack.toString());
                } else {
                    throw new NotSerializableException(cl.getName());
                }
            }
        } finally {
            depth--;
            bout.setBlockDataMode(oldMode);
```

```
ObjectOutputStream › writeObject0()
```

```
java.exe ...

eption: cn.User
tStream.writeObject0(ObjectOutputStream.java:1184)
tStream.writeObject(ObjectOutputStream.java:348)
```

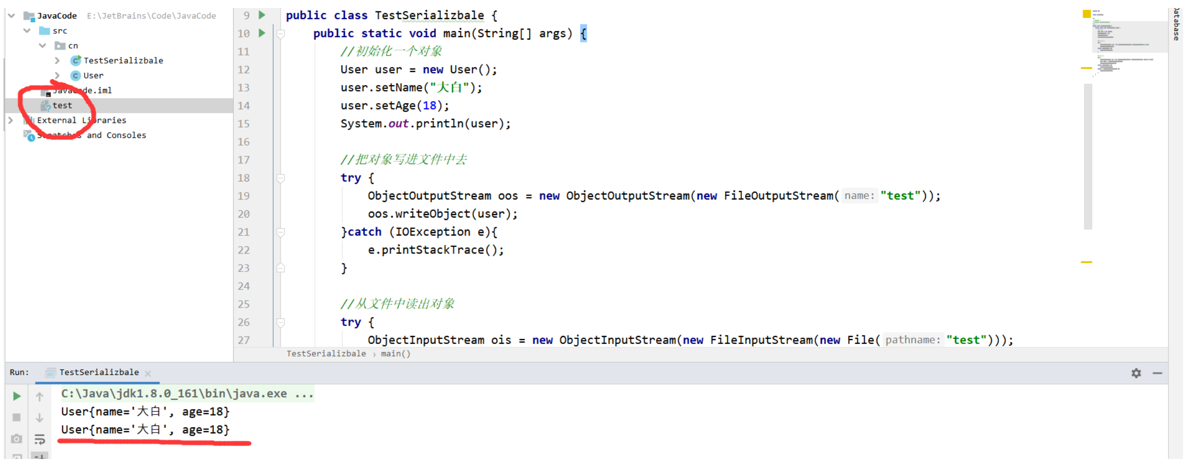
可以看到，最后是判断了当前对象，是不是 Serializable 这个接口的实现类。如果不是就会抛出 NotSerializableException 这个异常。

接下来，我们实现 Serializable 之后



```java
public class User implements Serializable {
    private String name;
    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }
```

运行结果:



```
JavaCode  E:\JetBrains\Code\JavaCode
  src
    cn
      TestSerializbale
      User
  JavaCode.iml
  test
External Libraries
Scratches and Consoles
```

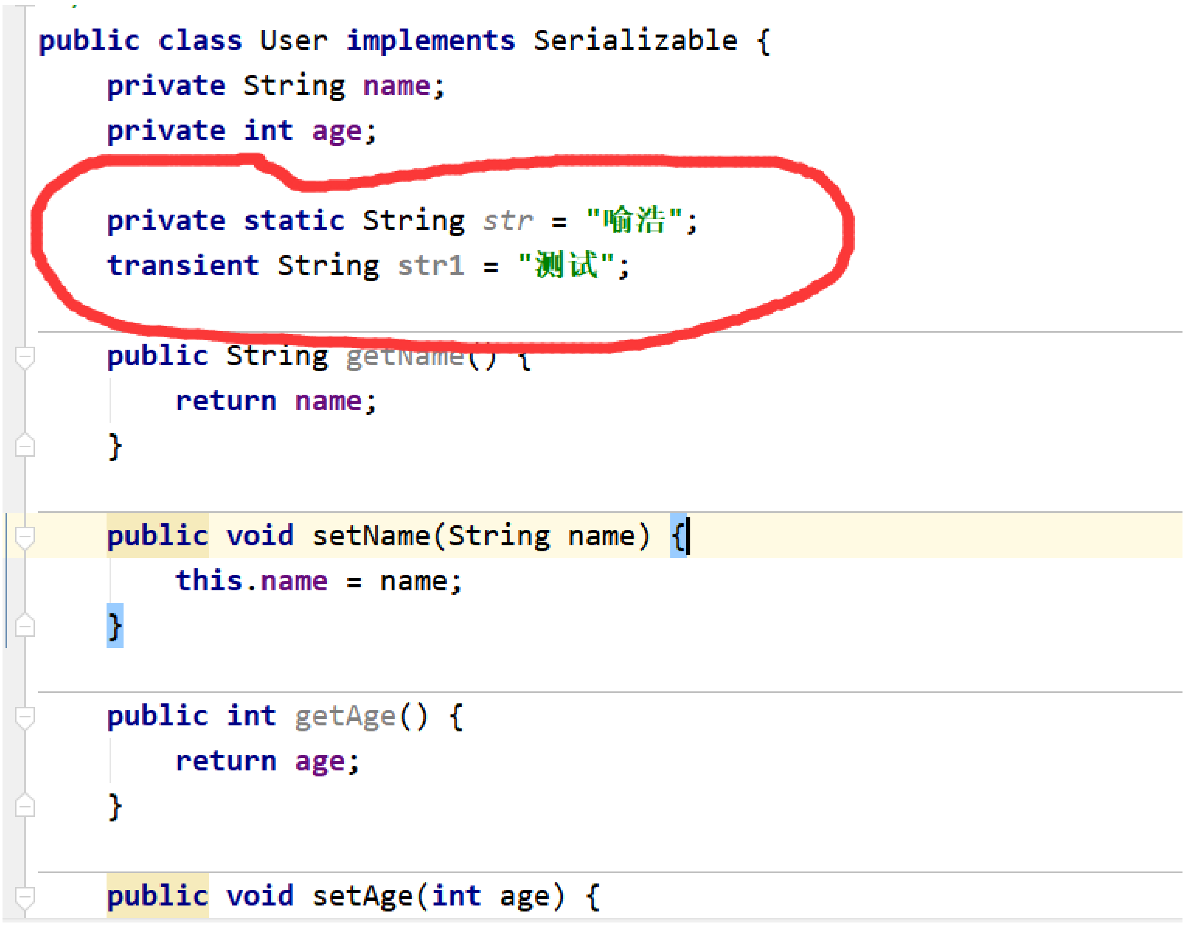
```java
public class TestSerializbale {
    public static void main(String[] args) {
        //初始化一个对象
        User user = new User();
        user.setName("大白");
        user.setAge(18);
        System.out.println(user);

        //把对象写进文件中去
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
```

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
User{name='大白', age=18}
User{name='大白', age=18}
```

可以看到，我们成功的将对象存储到文件中后，又将其取了出来

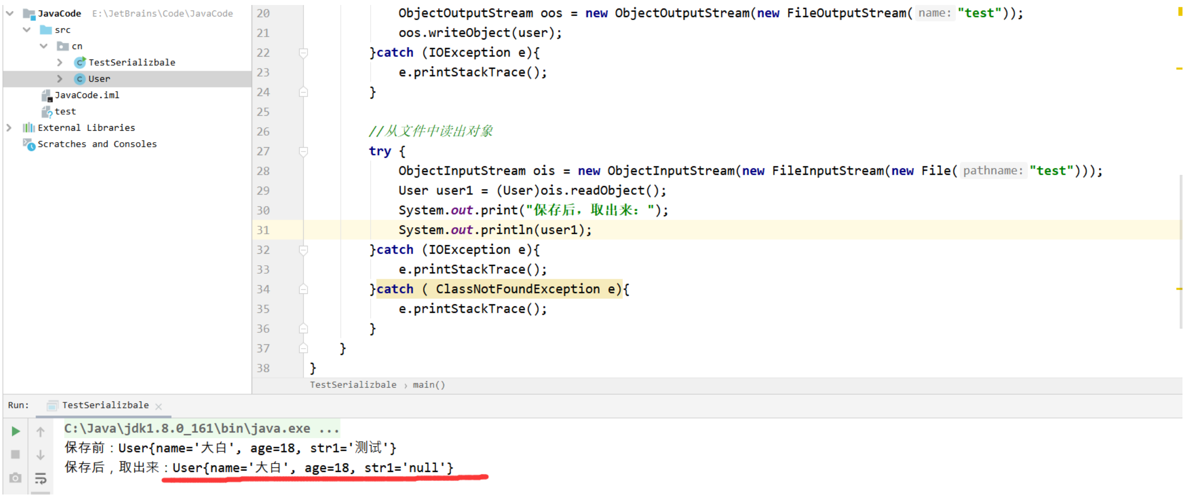
static和transient关键字

添加 static 和transient 修饰的变量



```java
public class User implements Serializable {
    private String name;
    private int age;

    private static String str = "喻浩";
    transient String str1 = "测试";

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
```

再次测试方法：

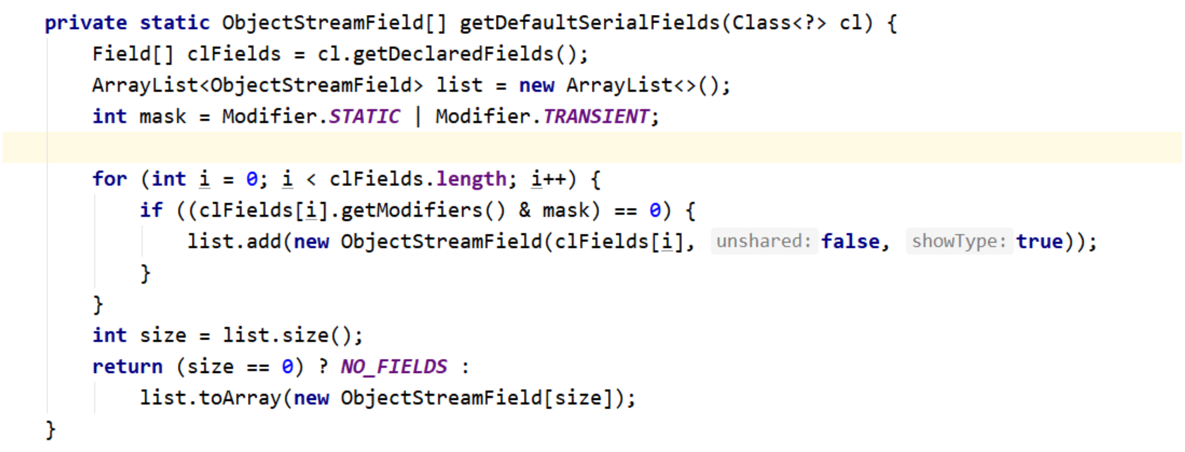


```
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User user1 = (User)ois.readObject();
            System.out.print("保存后，取出来：");
            System.out.println(user1);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

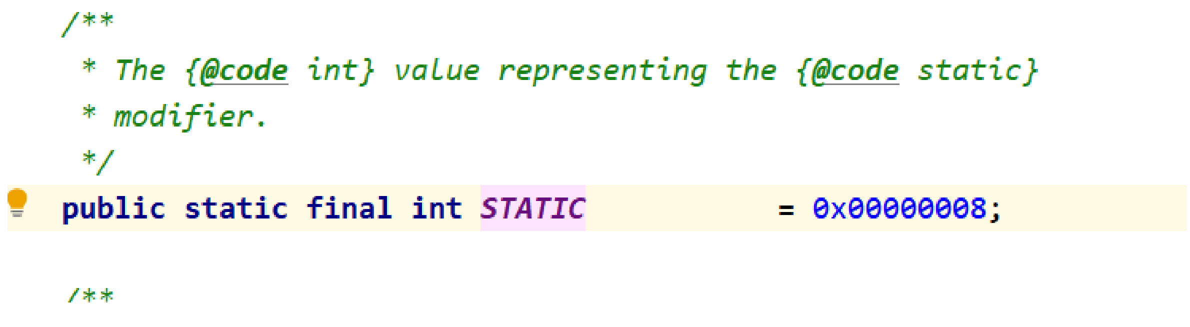
```
C:\Java\jdk1.8.0_161\bin\java.exe ...
保存前：User{name='大白', age=18, str1='测试'}
保存后，取出来：User{name='大白', age=18, str1='null'}
```

可以看到，transient 修饰的属性，在保存的时候并没有将其值保存下来。（在反序列化之后，会将其修饰的属性，变为默认值）

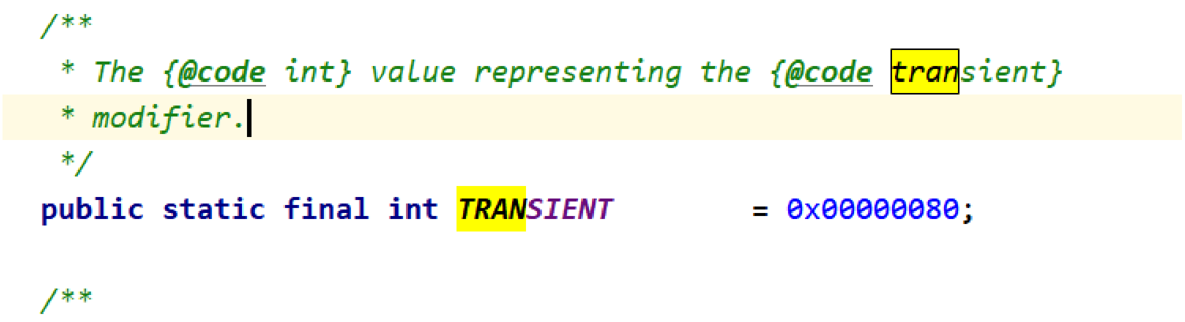
而 static 修饰的字段 属于类的状态，而序列化则是保存对象的状态，所以，序列化并不会保存 static 修饰的字段。



```
private static ObjectStreamField[] getDefaultSerialFields(Class<?> cl) {
    Field[] clFields = cl.getDeclaredFields();
    ArrayList<ObjectStreamField> list = new ArrayList<>();
    int mask = Modifier.STATIC | Modifier.TRANSIENT;

    for (int i = 0; i < clFields.length; i++) {
        if ((clFields[i].getModifiers() & mask) == 0) {
            list.add(new ObjectStreamField(clFields[i],  unshared: false,  showType: true));
        }
    }
    int size = list.size();
    return (size == 0) ? NO_FIELDS :
        list.toArray(new ObjectStreamField[size]);
}
```

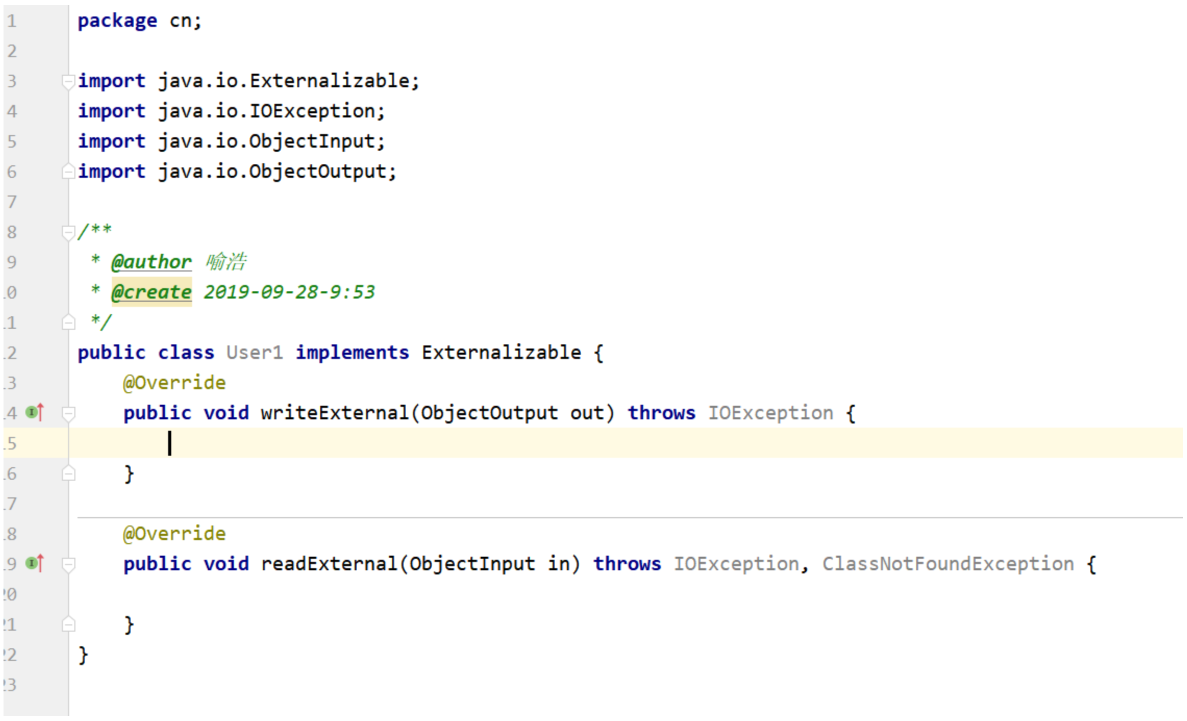


```
 /**
  * The {@code int} value representing the {@code static}
  * modifier.
  */
 public static final int STATIC           = 0x00000008;

 /**
```



```
/**
 * The {@code int} value representing the {@code transient}
 * modifier.
 */
public static final int TRANSIENT        = 0x00000080;

/**
```

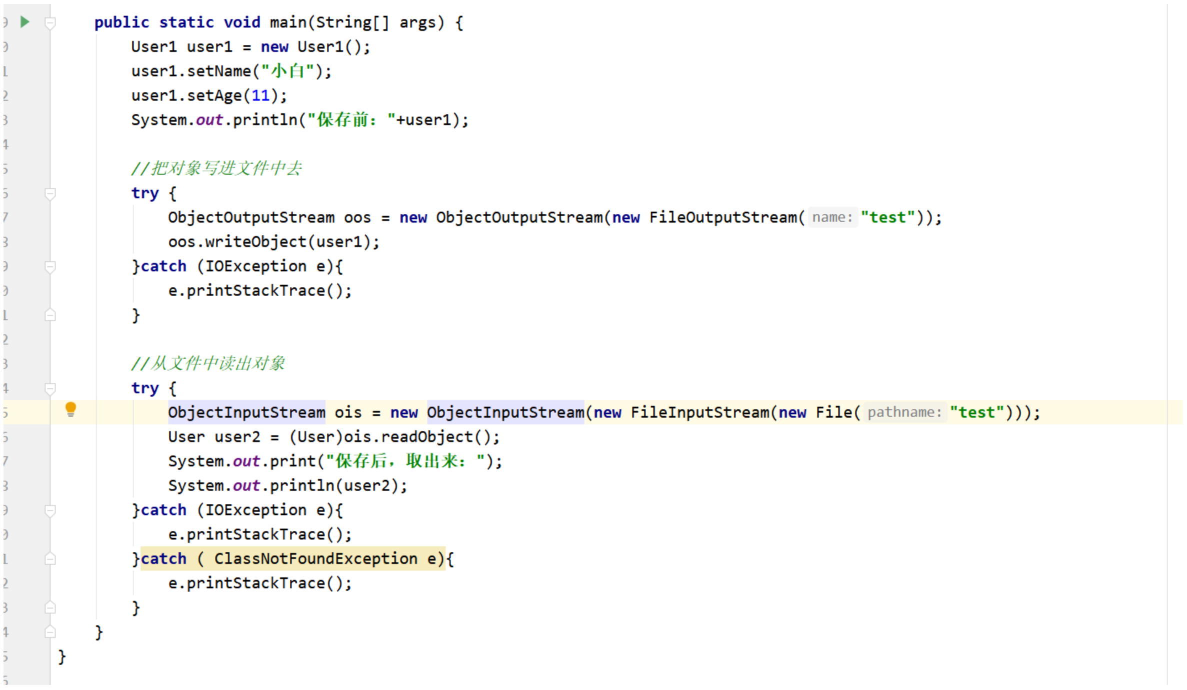
Externalizable接口



```java
package cn;

import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

/**
 * @author 喻浩
 * @create 2019-09-28-9:53
 */
public class User1 implements Externalizable {
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {

    }
}
```

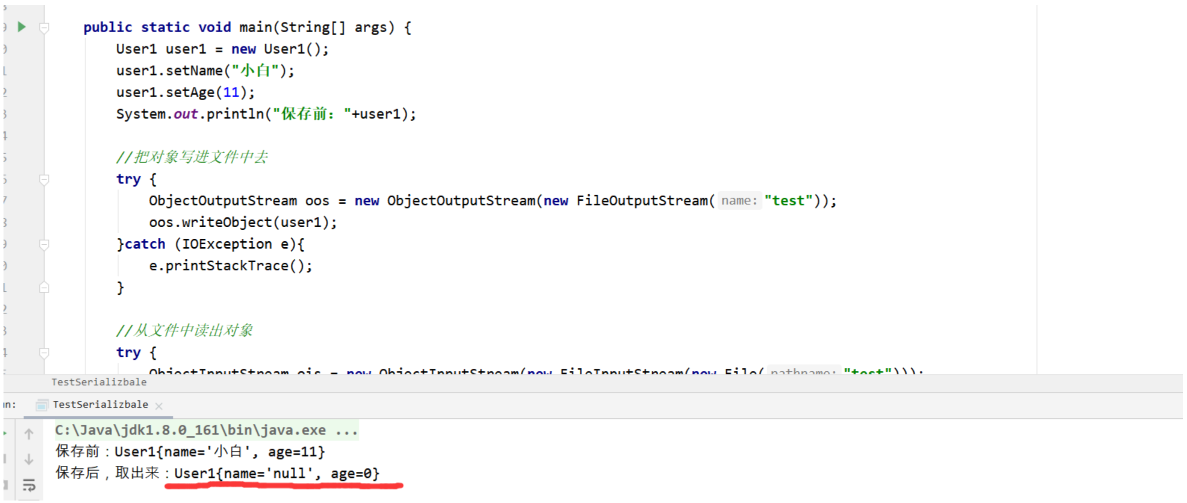
序列化还可以选择实现 Externalizable接口，但是必须要实现 writeExternal 和 readExternal 方法。

接下来，我们再次测试序列化。



```java
    public static void main(String[] args) {
        User1 user1 = new User1();
        user1.setName("小白");
        user1.setAge(11);
        System.out.println("保存前："+user1);

        //把对象写进文件中去
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user1);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User user2 = (User)ois.readObject();
            System.out.print("保存后，取出来：");
            System.out.println(user2);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

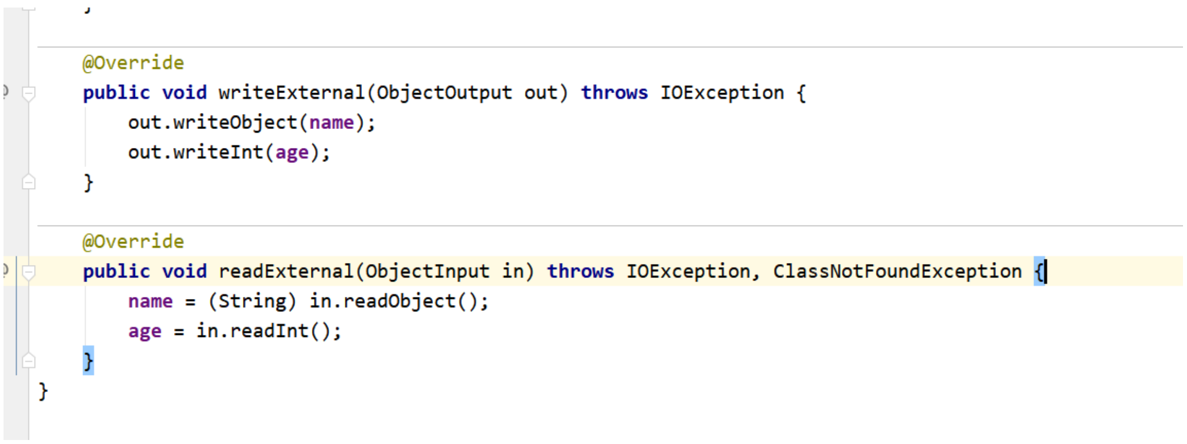
可以看到结果是：



```
public static void main(String[] args) {
    User1 user1 = new User1();
    user1.setName("小白");
    user1.setAge(11);
    System.out.println("保存前: "+user1);

    //把对象写进文件中去
    try {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
        oos.writeObject(user1);
    }catch (IOException e){
        e.printStackTrace();
    }

    //从文件中读出对象
    try {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
```

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
保存前：User1{name='小白', age=11}
保存后，取出来：User1{name='null', age=0}
```

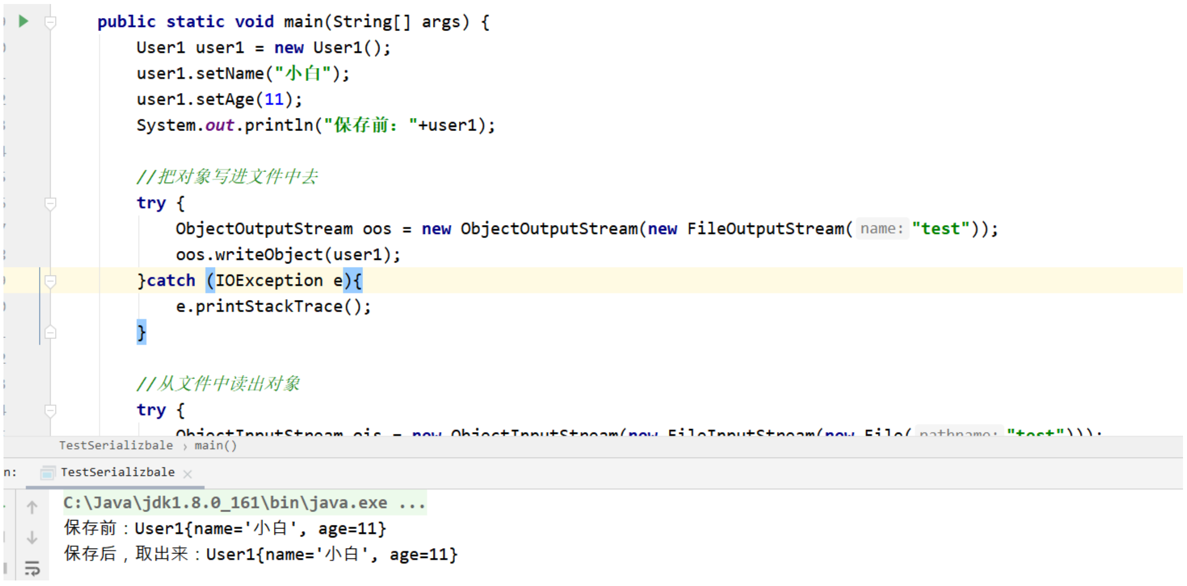
原因是，我们必须要重写 Externalizable接口 让我们实现的方法



```
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        out.writeObject(name);
        out.writeInt(age);
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        name = (String) in.readObject();
        age = in.readInt();
    }
}
```

就是这样。

再次测试：



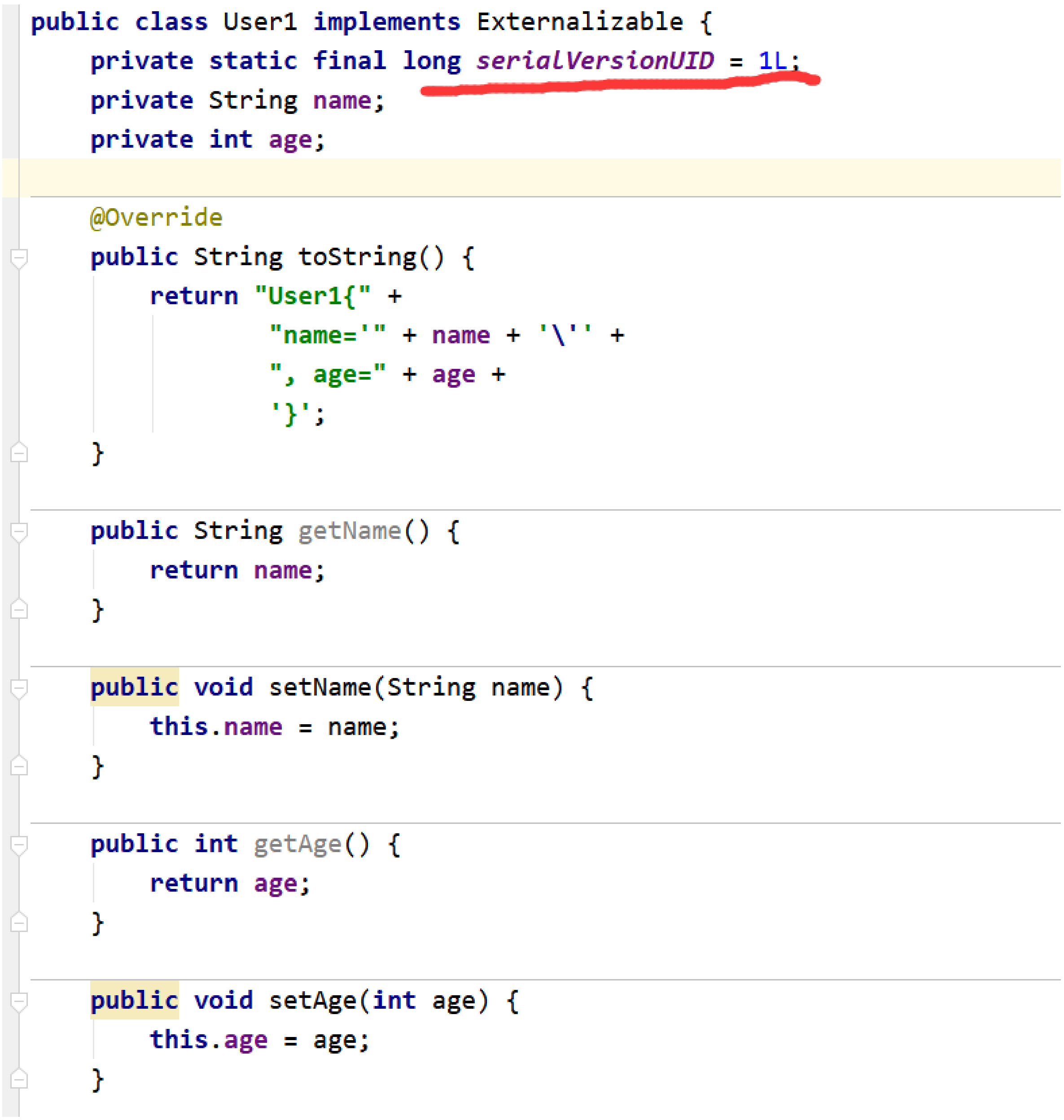
```
public static void main(String[] args) {
    User1 user1 = new User1();
    user1.setName("小白");
    user1.setAge(11);
    System.out.println("保存前: "+user1);

    //把对象写进文件中去
    try {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
        oos.writeObject(user1);
    }catch (IOException e){
        e.printStackTrace();
    }

    //从文件中读出对象
    try {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
```

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
保存前：User1{name='小白', age=11}
保存后，取出来：User1{name='小白', age=11}
```

结果正常

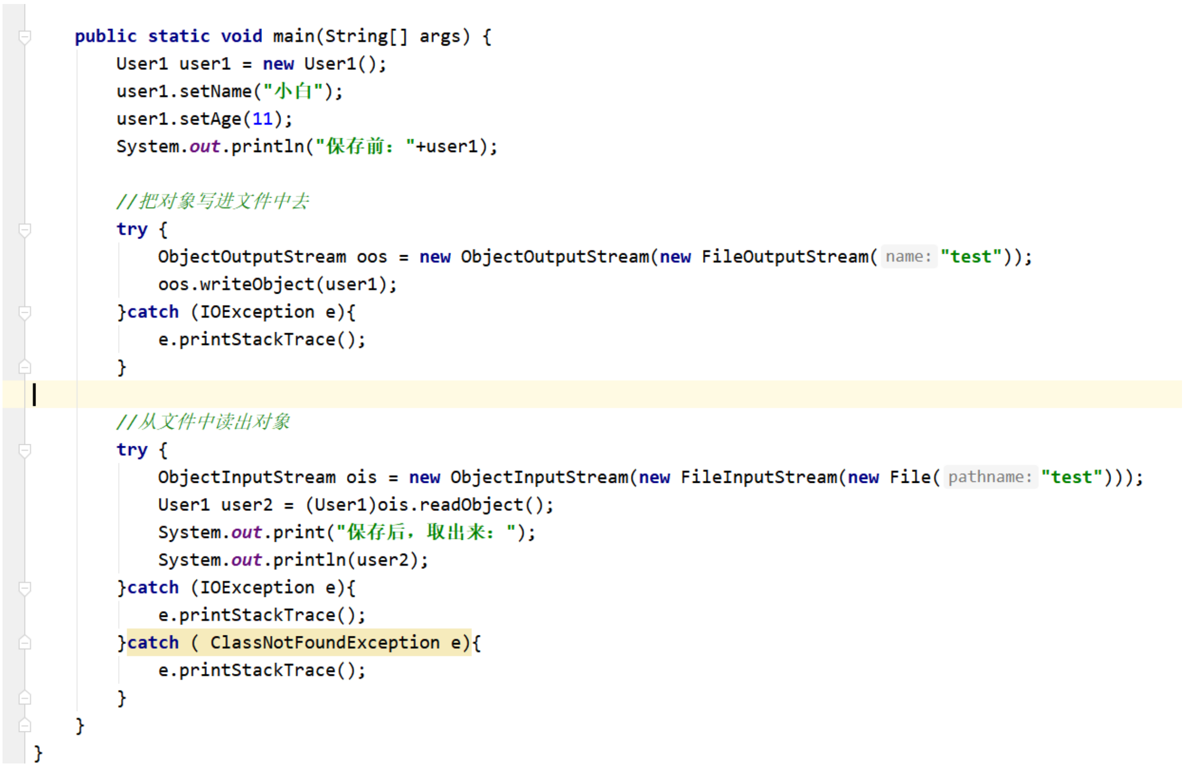
serialVersionUID

首先，给对象 加一条属性

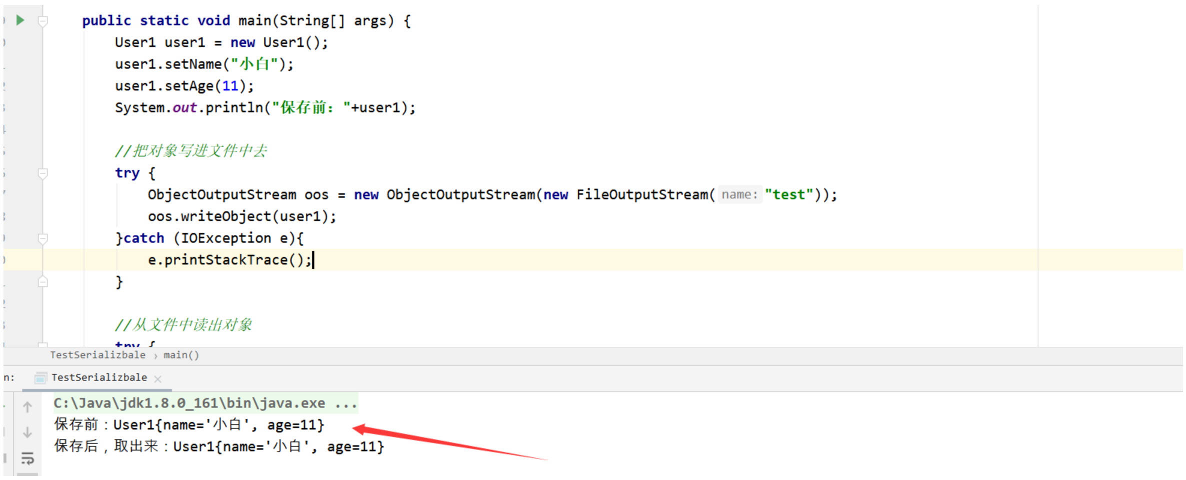


```java
public class User1 implements Externalizable {
    private static final long serialVersionUID = 1L;
    private String name;
    private int age;

    @Override
    public String toString() {
        return "User1{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

然后，运行测试方法



```java
    public static void main(String[] args) {
        User1 user1 = new User1();
        user1.setName("小白");
        user1.setAge(11);
        System.out.println("保存前: "+user1);

        //把对象写进文件中去
        try {
            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
            oos.writeObject(user1);
        }catch (IOException e){
            e.printStackTrace();
        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User1 user2 = (User1)ois.readObject();
            System.out.print("保存后，取出来: ");
            System.out.println(user2);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```



```java
public static void main(String[] args) {
    User1 user1 = new User1();
    user1.setName("小白");
    user1.setAge(11);
    System.out.println("保存前: "+user1);

    //把对象写进文件中去
    try {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "test"));
        oos.writeObject(user1);
    }catch (IOException e){
        e.printStackTrace();
    }

    //从文件中读出对象
```

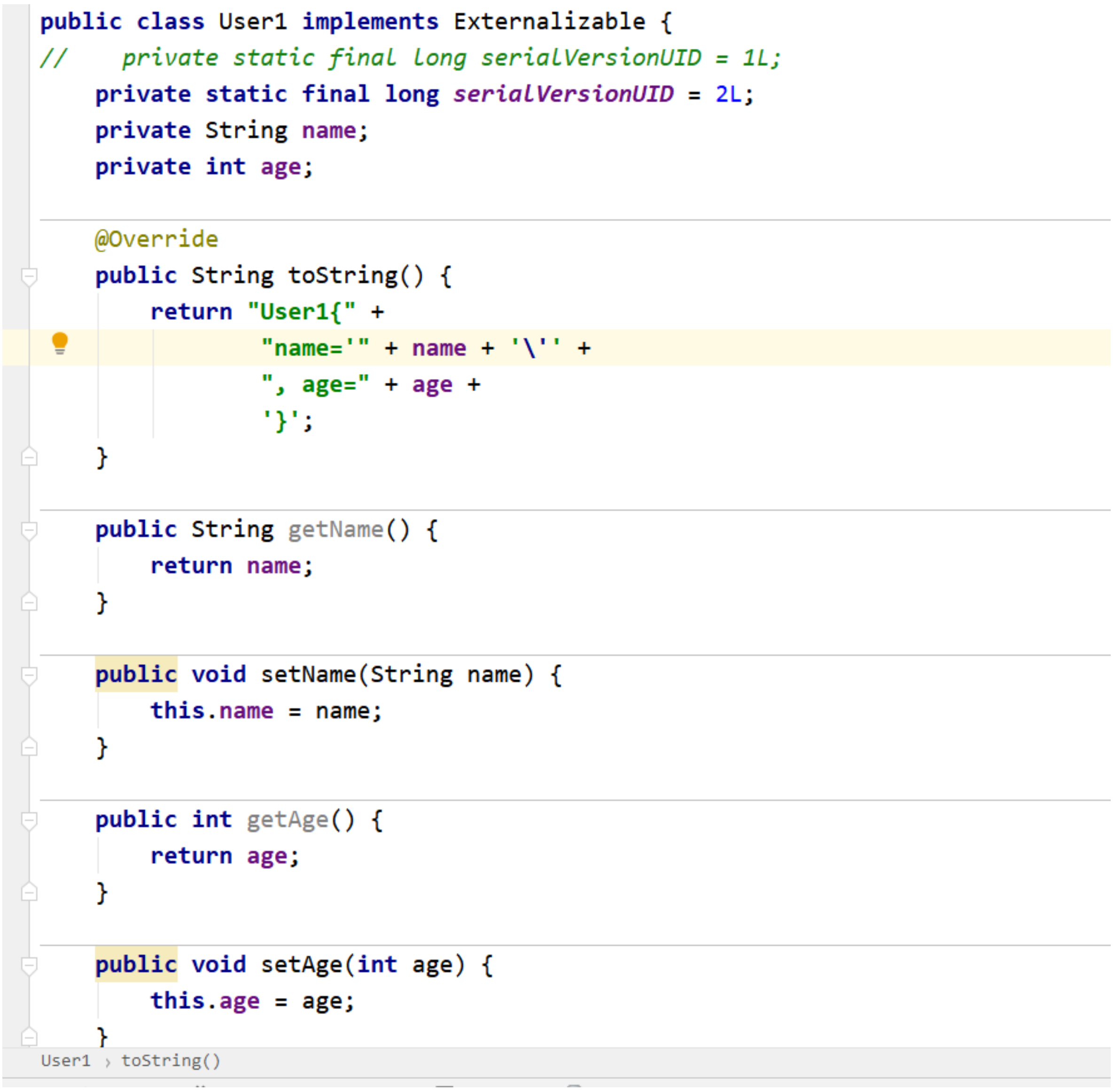
TestSerializbale › main()

TestSerializbale

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
保存前：User1{name='小白', age=11}
保存后，取出来：User1{name='小白', age=11}
```

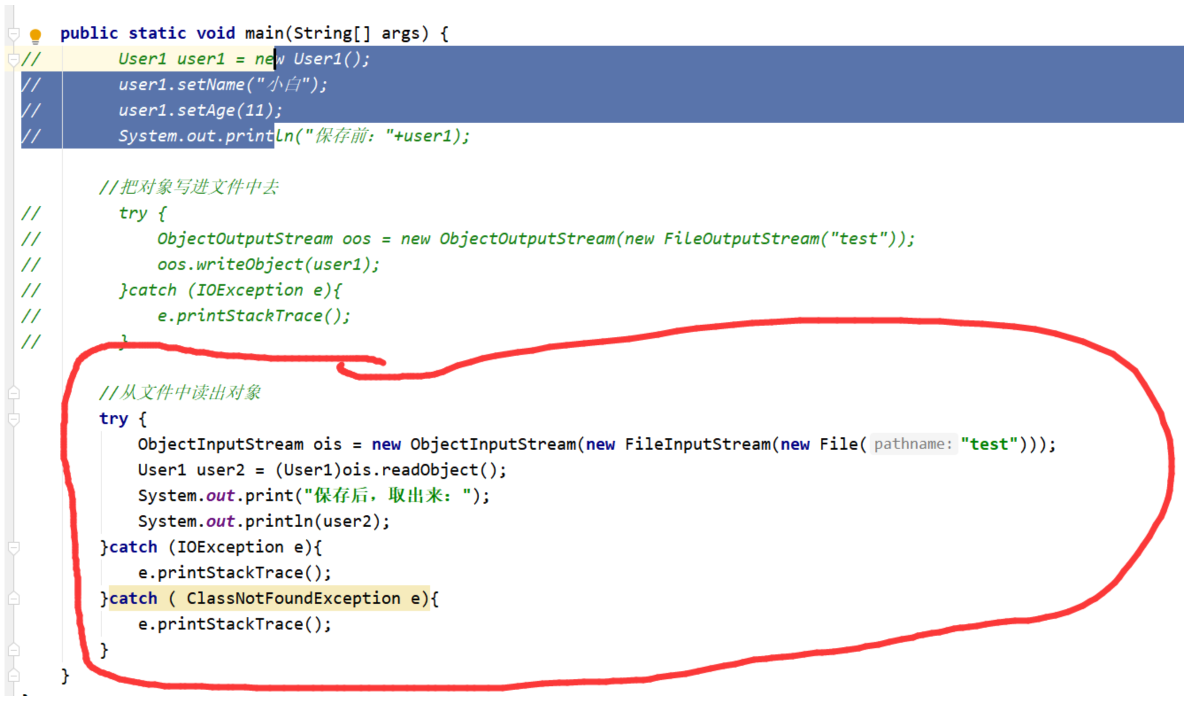
结果如上。

这时，我们把 User1 类里面的 serialVersionUID 进行修改。进行反序列化操作



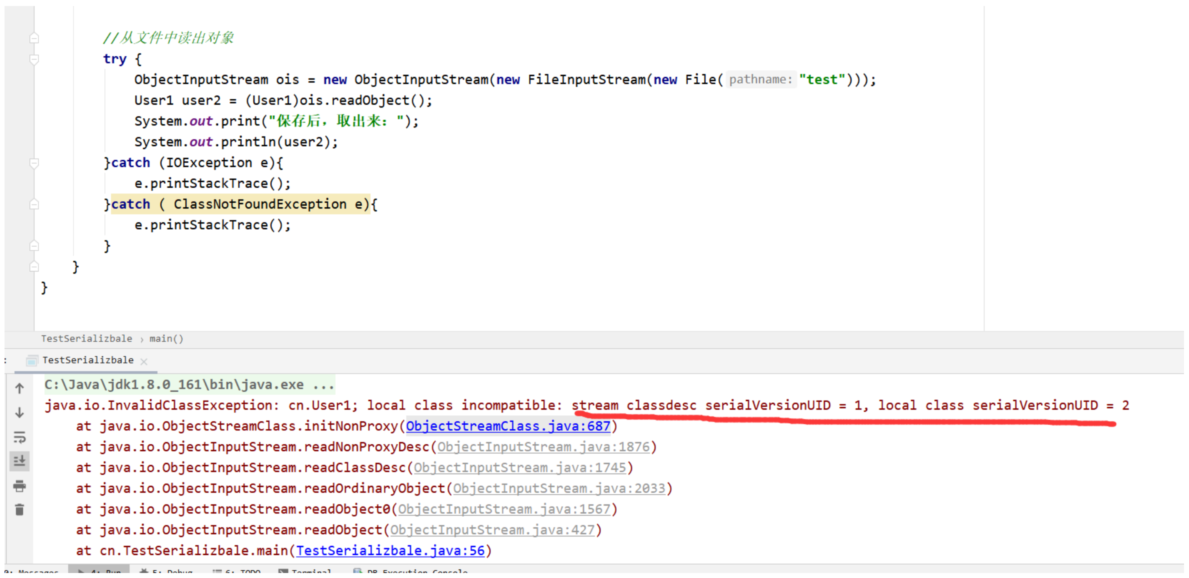
```java
public class User1 implements Externalizable {
//    private static final long serialVersionUID = 1L;
    private static final long serialVersionUID = 2L;
    private String name;
    private int age;

    @Override
    public String toString() {
        return "User1{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

User1 › toString()



```java
    public static void main(String[] args) {
//        User1 user1 = new User1();
//        user1.setName("小白");
//        user1.setAge(11);
//        System.out.println("保存前: "+user1);

        //把对象写进文件中去
//        try {
//            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("test"));
//            oos.writeObject(user1);
//        }catch (IOException e){
//            e.printStackTrace();
//        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User1 user2 = (User1)ois.readObject();
            System.out.print("保存后，取出来: ");
            System.out.println(user2);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
```

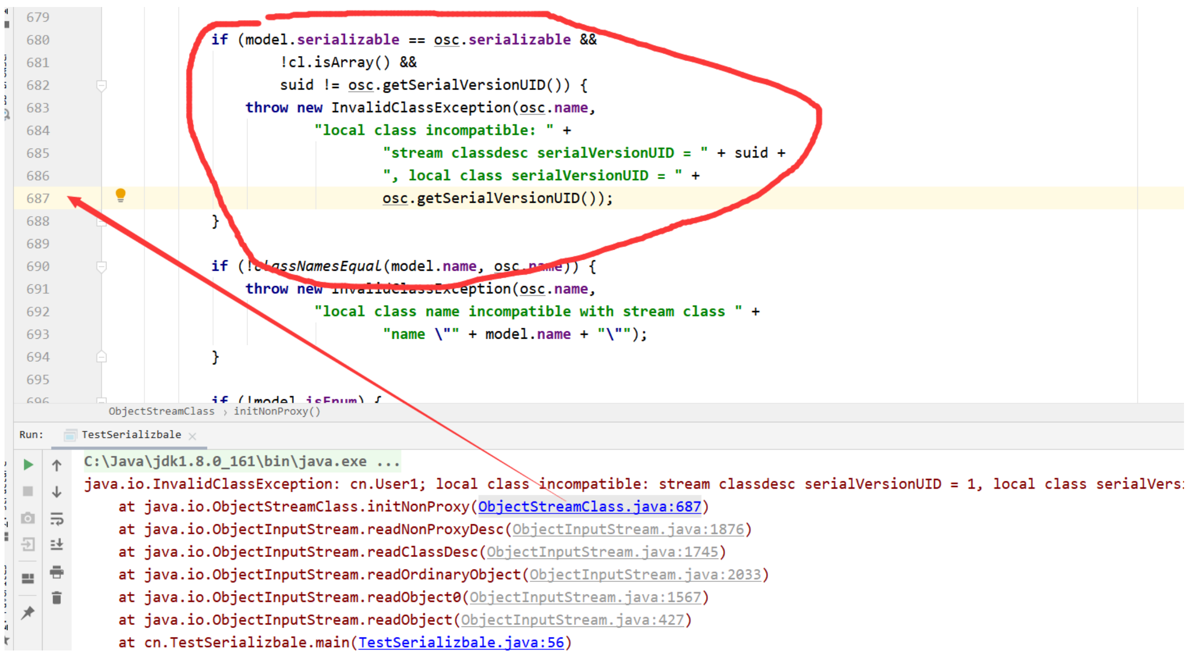
报错显示 serialVersionUID 不一致：



```java
        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User1 user2 = (User1)ois.readObject();
            System.out.print("保存后，取出来: ");
            System.out.println(user2);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
}
```

```
TestSerializbale › main()
TestSerializbale ×
C:\Java\jdk1.8.0_161\bin\java.exe ...
java.io.InvalidClassException: cn.User1; local class incompatible: stream classdesc serialVersionUID = 1, local class serialVersionUID = 2
    at java.io.ObjectStreamClass.initNonProxy(ObjectStreamClass.java:687)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1876)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1745)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2033)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1567)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:427)
    at cn.TestSerializbale.main(TestSerializbale.java:56)
```

跟进源码：



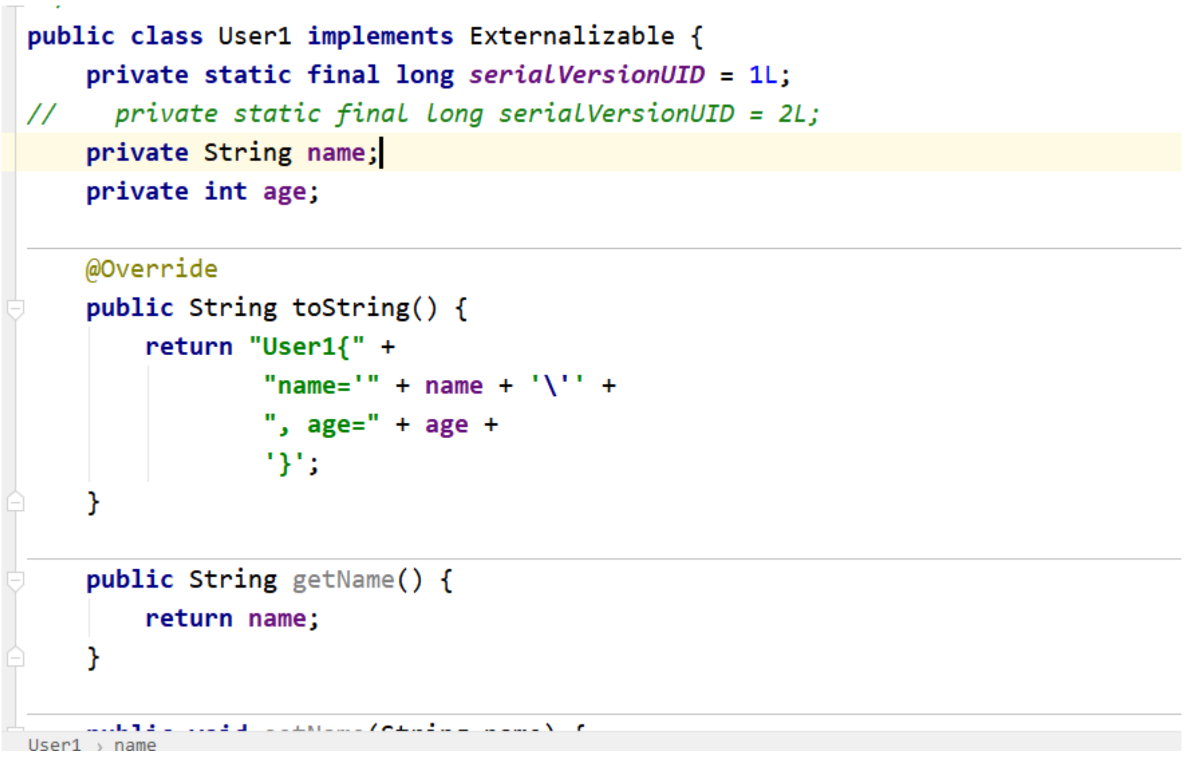
```java
679
680             if (model.serializable == osc.serializable &&
681                     !cl.isArray() &&
682                     suid != osc.getSerialVersionUID()) {
683                 throw new InvalidClassException(osc.name,
684                         "local class incompatible: " +
685                                 "stream classdesc serialVersionUID = " + suid +
686                                 ", local class serialVersionUID = " +
687                                 osc.getSerialVersionUID());
688             }
689
690             if (!classNamesEqual(model.name, osc.name)) {
691                 throw new InvalidClassException(osc.name,
692                         "local class name incompatible with stream class " +
693                                 "name \"" + model.name + "\"");
694             }
695
696             if (!model.isEnum) {
```

```
ObjectStreamClass › initNonProxy()
Run:    TestSerializbale ×
C:\Java\jdk1.8.0_161\bin\java.exe ...
java.io.InvalidClassException: cn.User1; local class incompatible: stream classdesc serialVersionUID = 1, local class serialVers
    at java.io.ObjectStreamClass.initNonProxy(ObjectStreamClass.java:687)
    at java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1876)
    at java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1745)
    at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2033)
    at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1567)
    at java.io.ObjectInputStream.readObject(ObjectInputStream.java:427)
    at cn.TestSerializbale.main(TestSerializbale.java:56)
```

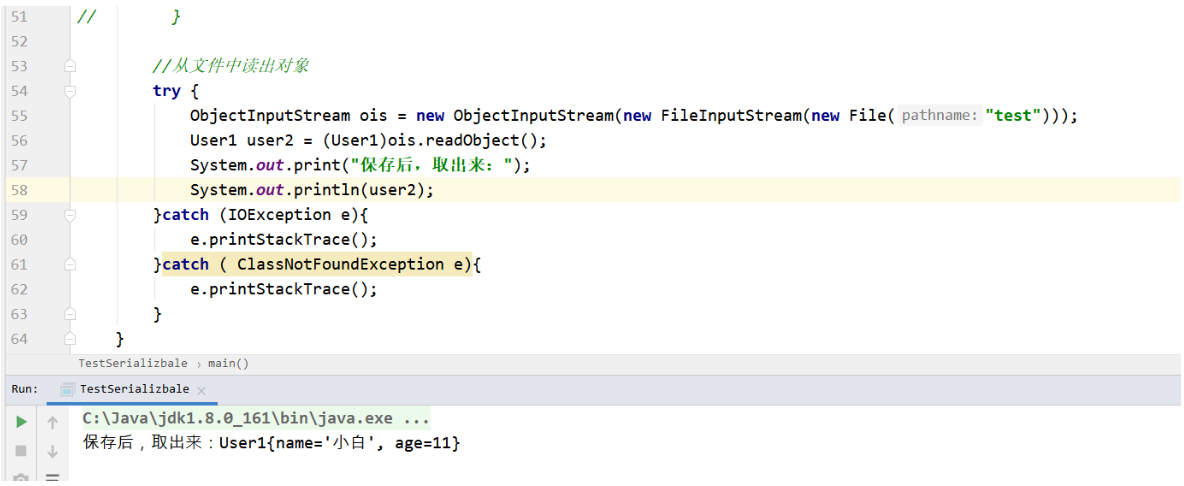
大概意思就是，在反序列化的时候，会比较文件中的 serialVersionUID 和当前类中的 serialVersionUID 如果不一致，则会抛出 stream classdesc serialVersionUID = 1, local class serialVersionUID = 2 异常

如果我们不对类的 serialVersionUID 进行设置，JVM会使用自己的算法生成默认的 serialVersionUID 。

将 serialVersionUID 改回后：



```java
public class User1 implements Externalizable {
    private static final long serialVersionUID = 1L;
//    private static final long serialVersionUID = 2L;
    private String name;
    private int age;

    @Override
    public String toString() {
        return "User1{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    public String getName() {
        return name;
    }
```



```java
//        }

        //从文件中读出对象
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File( pathname: "test")));
            User1 user2 = (User1)ois.readObject();
            System.out.print("保存后，取出来：");
            System.out.println(user2);
        }catch (IOException e){
            e.printStackTrace();
        }catch ( ClassNotFoundException e){
            e.printStackTrace();
        }
    }
```

```
C:\Java\jdk1.8.0_161\bin\java.exe ...
保存后，取出来：User1{name='小白', age=11}
```

和上文一样