1 JDK 定义的 Serializable 接口

在 java.io.Serializable 接口定义为：

```
public interface Serializable {
}
```

注意 java.io.Serializable 接口是一个空接口，里面没有任何方法。java.io.Serializable 接口的一个例子如下所示：

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

    //注意这里没有缺省构造
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String toString() {
        return "name = " + name + ", age = " + age;
    }

    public static void main(String[] args) {
        //序列化对象到文件
        try (//创建一个 ObjectOutputStream 输出流
             //写到二进制文件，因此 d:/object.txt 打开都是乱码
             ObjectOutputStream oos = new ObjectOutputStream(
                     new FileOutputStream("d:/object.txt"))) {
            //将对象序列化到文件
            Person person = new Person("aaa", 23);
            oos.writeObject(person);
        } catch (Exception e) {
            e.printStackTrace();
        }

        //然后从 d:/object.txt 文件里反序列化，直接产生新的对象
        try{
            ObjectInputStream ois = new ObjectInputStream(
                    new FileInputStream("d:/object.txt"));
            //注意这个时候不是调用构造函数，而是直接从文件反序列化到内存里产生对象
            Person newPerson = (Person) ois.readObject();
            System.out.println(newPerson);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

注意上面代码调用 ObjectOutputStream 的 writeObject 方法将对象序列化到文件，参数是被序列化的对象；调用 ObjectInputStream 的 readObject 方法从文件反序列化出对象，不带参数，返回 Person 类型对象。

序列化机制允许将实现序列化的 Java 对象转换位字节序列，这些字节序列可以保存在磁盘上，或通过网络传输，以达到目的主机的目标进程，以后通过反序列化恢复成原来的对象。序列化机制使得对象可以脱离程序的运行而独立存在。**序列化和反序列化不需要我们去关心文件的存储格式，特别是反序列化时不需要管对象的每个属性内容保存的格式。**

2 实验一所定义的 FileSerializable 接口

FileSerializable 接口定义为：

```
/**
 * 定义文件序列化接口
 */
public interface FileSerializable extends java.io.Serializable{
    /**
     * 写到二进制文件
     * @param out :输出流对象
     */
    public abstract void writeObject(ObjectOutputStream out);

    /**
     * 从二进制文件读
     * @param in ：输入流对象
     */
    public  abstract void readObject(ObjectInputStream in);
}
```

由于定义了二个接口方法，方法参数分别是 ObjectOutputStream 和 ObjectInputStream 类型对象，**因此这二个接口方法必须通过实现该接口的类的实例去调用，因此如果定义 Person 类来实现 FileSerializable 接口，那么实现方式和前面的例子是有区别的，特别是 Person 类需要定义缺省构造函数。下面是例子：**

```java
public class Person1 implements FileSerializable {
    private String name;
    private int age;

    //需要定义一个缺省构造函数，因为现在 Person1 实现接口 FileSerializable
    //必须通过对象调用 FileSerializable 方法
    public Person1(){}

    public Person1(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public void writeObject(ObjectOutputStream out) {
        try {
            //将 this 对象的成员依次序列化
            out.writeObject(this.name);
            out.writeObject(this.age);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void readObject(ObjectInputStream in) {
        //将 this 对象的成员依次反序列化，注意和序列化次序要一致
        try {
            this.name = (String)(in.readObject());
            this.age = (int)(in.readObject());
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public String toString() {
        return "name = " + name + ", age = " + age;
    }

    public static void main(String[] args){
        try {
            ObjectOutputStream out = new ObjectOutputStream(
                    new FileOutputStream("d:\\object.txt")) ;
            //将对象序列化到文件
            Person1 person = new Person1("aaa", 23);
            person.writeObject(out);  //注意现在是通过对象调用 writeObject，参数是 out
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
        //反序列化
        try {
            ObjectInputStream in = new ObjectInputStream(
                    new FileInputStream("d:\\object.txt"));
            //先构造一个空的 Person1 对象
            Person1 newPerson= new Person1();
            //再从文件反序列化出来对象所有成员
            newPerson.readObject(in);
            System.out.println(newPerson);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

请大家注意二个例子的区别。

3 实验一的序列化要求

1. 题目要求

实现一个基于内存的英文全文检索搜索引擎，需要完成以下功能：

功能 1：将指定目录下的一批.txt 格式的文本文件扫描并在内存里建立倒排索引，这里面包含必须的子功能包括：

（1）读取文本文件的内容；

（2）将内容切分成一个个的单词；

（3）过滤掉其中一些不需要的单词,例如数字、停用词（the, is and 这样的单词）、过短或过长的单词（例如长度小于 3 或长度大于 20 的单词）；

（4）利用 Java 的集合类在内存里建立过滤后剩下单词的倒排索引；

（5）内存里建立好的索引对象可以序列化到文件，同时可以从文件里反序列化成内存里的索引对象；

首先在实验一的实验报告里功能 1 的（5）要求为：内存里建立好的索引对象可以序列化到文件，同时可以从文件里反序列化成内存里的索引对象；**这是指序列化功能。**

除了以上功能上的要求外，其他要求包括：

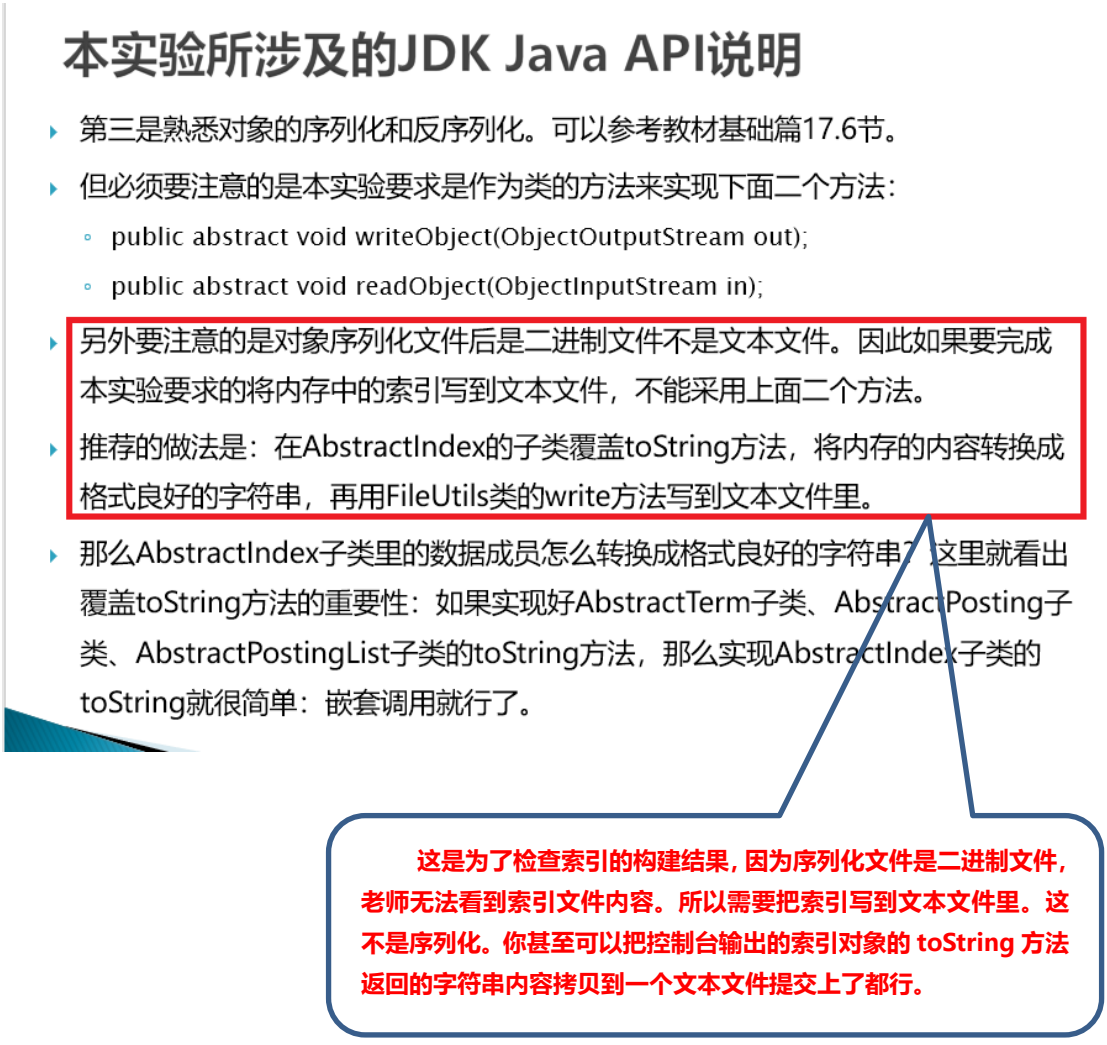
（1）针对搜索引擎的倒排索引结构，已经定义好了创建索引和全文检索所需要的抽象类和接口。**学生必须继承这些预定义的抽象类和和实现预定义接口来完成实验的功能，不能修改抽象类和接口里规定好的数据成员、抽象方法；也不能在预定义抽象类和接口里添加自己新的数据成员和方法。**但是实现自己的子类和接口实现类则不作任何限定。

（2）自己实现的抽象类子类和接口实现类里的关键代码必须加上注释，其中每个类、每个类里的公有方法要加上 Javadoc 注释，并自动生成 Java API 文档作为实验报告附件提交。

（3）使用统一的测试文档集合、统一的搜索测试案例对代码进行功能测试，构建好的索引和基于统一的搜索测试案例的检索结果最后输出到文本文件里作为实验报告附件提交。

实验一其他功能要求里的（3）为：使用统一的测试文档集合、统一的搜索测试案例对代码进行功能测试，构建好的索引和基于统一的搜索测试案例的检索结果最后输出到文本文件里作为实验报告附件提交。这是为了检查索引的构建结果，因为序列化文件是二进制文件，老师无法看到索引文件内容。所以需要把索引写到文本文件里。**这不是序列化。你甚至可以把控制台输出的索引对象的 toString 方法返回的字符串内容拷贝到一个文本文件提交上了都行。**

在实验一 PPT 里最后一页的解释同上面。



这是为了检查索引的构建结果，因为序列化文件是二进制文件，老师无法看到索引文件内容。所以需要把索引写到文本文件里。这不是序列化。你甚至可以把控制台输出的索引对象的 toString 方法返回的字符串内容拷贝到一个文本文件提交上了都行。