Java2 Lab 12
(Java I/o)

[Experimental Objective]

1. Learn to understand Java's Io operations
2. Master JAVA serialization and deserialization methods by writing and debugging programs

[Java I/O]

Java's input/output streams are mainly classified into two categories according to their data types: Character Stream and Bite Stream.

(1) character stream

The input/output data of the character stream is a character code, that is, a Unicode character. There are two basic classes for character streams: the Reader class and the Writer class.

Reader: An abstract class that cannot be used directly, and gets characters from a file through its subclass, FileReader.

Writer: An abstract class that cannot be used directly, writes characters to a file through its subclass, FileWriter.

(2) byte stream

The byte stream is read/write binary data in bytes. There are two basic classes for byte streams: the InputStream class and the OutputStream class.

InputStream: An abstract class that cannot be used directly, and gets its bytes from a file through its subclass FileInputStream.

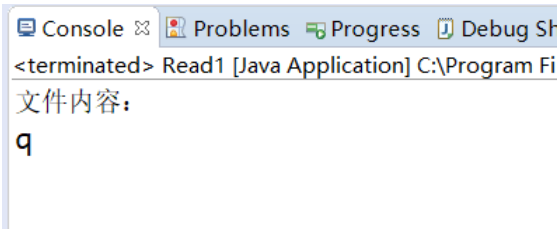
OutputStream: An abstract class that cannot be used directly, and writes bytes to the file through its subclass FileOutputStream.

[Case 1 character stream]

Use character stream, mainly use FileReader, read the information in the file, you can get a character, and the value is ASCII code type (int), and need to be converted to become char type. Examples are as follows:•

- **public class** Read1 {

```
    public static void main(String[] args) {
        try {
            FileReader fr = new FileReader("File.txt");
            System.out.print("文件内容：\n");
            int s = fr.read();    //读取到的字符的 ASCII 值，返回值是 int
            //ASCII 值转成字符
            char ss = (char)s;
            System.out.print(ss);
            fr.close();
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```



[Exercise 1 character stream]

Although the character stream can only be processed character by character, we can write a loop to improve it. Please use the character stream (ie using FileReader) to read the contents of the first line of the document in batches.

[Case 2 byte stream]

The byte stream is read/write binary data in bytes. The following example reads the first line of the file (ie the first 6 characters). You will find that the is.read(buffer, 0, 6) changes 6 to Both 7 and 8 show the first line of data, because the 7th character is '\n' and the 8th is '\r'.

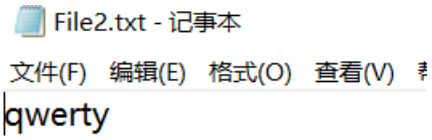
```
1.  public class Read2 {
2.
3.      public static void main(String[] args) {
4.          try {
5.              FileInputStream fis = new FileInputStream("File.txt");
6.              System.out.println("文档内容: ");
7.              byte[] buffer = new byte[1024];
8.              fis.read(buffer, 0, 6);
9.              System.out.println(new String(buffer));
10.             fis.close();
11.             System.out.println("reading finished");
12.         } catch (IOException e) {
13.             System.out.println(e);
```

```
        }
    }
}
```

```
Package Explorer  Type Hierarchy  Console
<terminated> Read2 [Java Application] C:\Program F
文档内容:
qwerty
reading finished
```

[Exercise 2 byte stream]

Please follow the case 2, using the byte stream method (that is, using FileOutputStream), batch write a string into the file.
effect:



[Case 3 Bufferstream]

The Java filter stream itself does not have IO functionality, but filters are added to other streams to improve efficiency, such as loading a package for other streams. Filtered streams are divided into byte and character filter streams
The byte filter stream is:
BufferedInputStream—字节输入过滤流
BufferedOutputStream—字节输出过滤流
Character filter stream is:
BufferedReader—字符输入过滤流
BufferedWriter—字符输出过滤流

If you copy a file, do not use the filter stream as follows:

```java
public class FileCopy {
    public static void main(String[] args) {
        try (FileInputStream in = new FileInputStream("./TestDir/src.zip");
                FileOutputStream out = new FileOutputStream("./TestDir/subDir/src.zip")) {
            //开始时间，当前系统纳秒时间
            long startTime = System.nanoTime();
            // 准备一个缓冲区
            byte[] buffer = new byte[1024];
            // 首先读取一次
            int len = in.read(buffer);
            while (len != -1) {
                // 开始写入数据
                out.write(buffer, 0, len);
                // 再读取一次
                len = in.read(buffer);
            }
            //结束时间，当前系统纳秒时间
            long elapsedTime = System.nanoTime() - startTime;
            System.out.println("耗时:" + (elapsedTime / 1000000.0) + " 毫秒");
        } catch (FileNotFoundException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
Package Explorer  Type Hierar
<terminated> FileCopy [Java Applic
耗时：3714.898417 毫秒
```

Use the filtered stream as follows:

```
public class FileCopyWithBuffer {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("./TestDir/src.zip");
                BufferedInputStream bis = new BufferedInputStream(fis);
                FileOutputStream fos = new FileOutputStream("./TestDir/subDir/src.zip");
                BufferedOutputStream bos = new BufferedOutputStream(fos)) {

            //开始时间
            long startTime = System.nanoTime();
            // 准备一个缓冲区
            byte[] buffer = new byte[1024];
            // 首先读取一次
            int len = bis.read(buffer);
            while (len != -1) {
                // 开始写入数据
                bos.write(buffer, 0, len);
                // 再读取一次
                len = bis.read(buffer);
            }
            //结束时间
            long elapsedTime = System.nanoTime() - startTime;
            System.out.println("耗时：" + (elapsedTime / 1000000.0) + " 毫秒");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
Package Explorer  Type Hierarchy
<terminated> FileCopyWithBuffer [Java App
耗时：45.812339 毫秒
```

It can be seen that the use of filtered streams significantly increases the speed of replication.

[Exercise 3 Bufferstream]

Please use the Bufferstream to read the entire contents of a txt file.

```
File.txt - 记事本
Package Explorer  Type Hierarchy  Co
<terminated> Read3_2 [Java Application] C:\P
文档内容：
qwerty
asdfgh
zxcvbn
qwewerq
reading finished
```

```
File.txt - 记事本
文件(F)  编辑(E)  格
qwerty
asdfgh
zxcvbn
qwewerq
```

【Serialization and deserialization in java】

Serialization: The process of converting a Java object into a sequence of bytes is called serialization of the object.

Deserialization: The process of restoring a sequence of bytes to a Java object is called deserialization of the object.

There are two main uses for serialization of objects:

1) Permanently save the byte sequence of the object to the hard disk, usually in a file;

2) Transfer the byte sequence of the object over the network.

Object serialization includes the following steps:

1) Create an object output stream that wraps a different type of target output stream, such as a file output stream;

2) Write the object through the writeObject() method of the object output stream.

[Case 4 Serialization]

First create a student class as follows, remember to define it as a class that can be serialized. Its members include name, age, student number, and test scores.

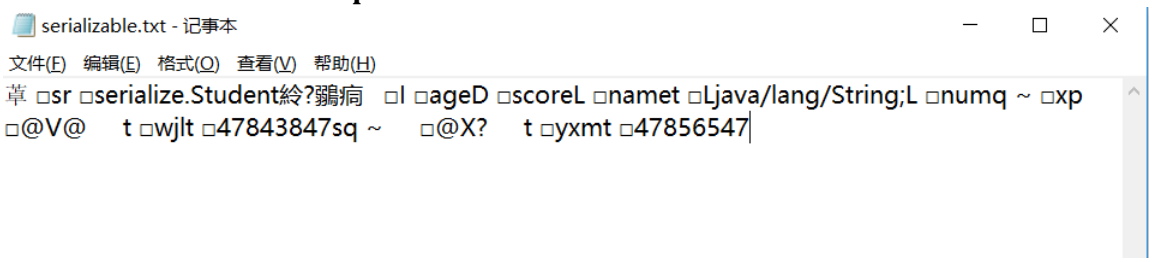
```
1.  class Student implements Serializable {
2.      String name;
3.      int age;
4.      String num;
5.      double score;
6.      public Student() {
7.      }
8.      public Student(String name, int age, String num, double score) {
9.          this.name = name;
10.         this.age = age;
11.         this.num = num;
12.         this.score = score;
13.     }
14.     public String toString() {
15.         return name + "\t" + age + "\t" + num + "\t" + score;
16.     }
17. }
```

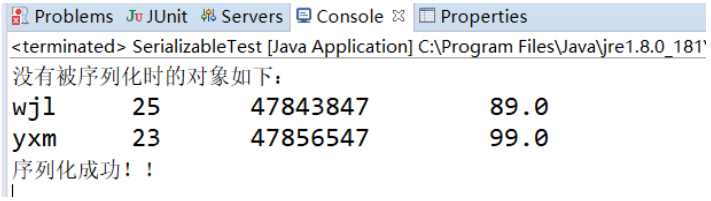
Then create two instances with the student class and write to the txt file:

```
1.      import java.io.File;
2.      import java.io.FileOutputStream;
```

```
3.  import java.io.IOException;
4.  import java.io.ObjectOutputStream;
5.  public class SerializableTest {
6.      public static void main(String[] args) {
7.          Student stu_1 = new Student("wjl", 25, "47843847", 89);
8.          Student stu_2 = new Student("yxm", 23, "47856547", 99);
9.          File f = new File("serializable.txt");
10.         try {
11.             FileOutputStream fos = new FileOutputStream(f);
12.             ObjectOutputStream oos = new ObjectOutputStream(fos);
13.             System.out.println("没有被序列化时的对象如下：");
14.             System.out.println(stu_1);
15.             System.out.println(stu_2);
16.             oos.writeObject(stu_1);
17.             oos.writeObject(stu_2);
18.             System.out.println("序列化成功！！");
19.             oos.flush();
20.             fos.close();
21.             oos.close();
22.         } catch (IOException e) {
23.             e.printStackTrace();
24.         }
25.     }
26. }
```

You can see that the output after serialization is as follows:





[Exercise 4 Deserialization]

In Case 4, we learned how to serialize objects. In this exercise, you need to read the txt file generated by Case 4 as an object instance and output the details to the console.

The effect is as follows: