讲义08：文件管理与输入输出流

華南農業大學
South China Agricultural University

主要内容

- 讲授内容
  - 使用File类对文件和目录进行管理
  - 字节流与字符流
  - 文件字节流与文件字符流
  - 缓冲流与数据流
  - 对象序列化与对象流
- 内容资料来源
  - 教材第10章
  - Java API文档

1 File类

- 为了能把数据长期保存，需要将它们存储到磁盘文件.
- 操作系统的文件系统中，文件位于一个目录.
- 绝对文件名是由驱动器字母、完全路径和文件名组成，它与操作系统有关，如windows中：
  C:\book\Welcome.java
- java.io.File类描述一个目录或文件对象.

1 File类

构造方法:

- File(String pathname)
  通过将给定路径名字符串转换成抽象路径名来创建一个新 File 实例。
- File(File parent, String child)
  根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。
- File(String parent, String child)
  根据 parent 路径名字符串和 child 路径名字符串创建一个新 File 实例。

1 File类

文件目录名相关的方法（均为public）

- String getName()
  返回由此抽象路径名表示的文件或目录的名称
- String getPath()
  将此抽象路径名转换为一个路径名字符串。
- String getAbsolutePath()
  返回抽象路径名的绝对路径名字符串。
- String getParent()
  返回此抽象路径名的父路径名的路径名字符串，如果此路径名没有指定父目录，则返回 null。

1 File类

文件目录属性相关的方法

- boolean exists()
- boolean canWrite()
- boolean canRead()
- boolean isDirectory()
- boolean isFile()
- boolean isHidden()

1 File类

与文件信息相关方法

- long lastModified(),
  返回此文件最后一次被修改的时间。
- long length(),
  返回由此文件的长度，字节为单位。

获取目录内容的方法

- File[] listFiles()
  返回目录中所有文件和子目录组成的数组。

1 File类

示例: 显示并统计一个目录中Java源程序文件。

① 输入一个目录名称, 如果该目录存在, 则完成以下步骤。

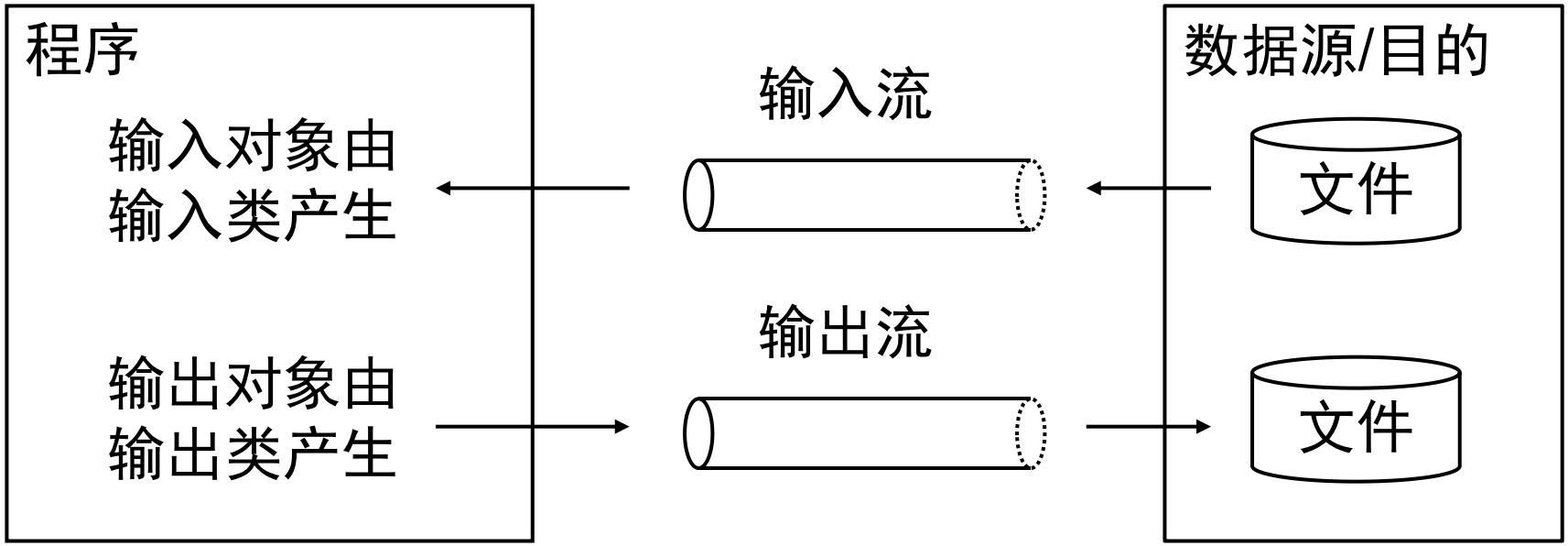
② 输出该目录中的所有Java源程序文件的名称、大小和最后修改时间。

③ 输出文件个数和文件总的大小。

示例代码：examples/Lecture08/JavaFileConuter

思考：如果不仅统计指定目录中的文件，还要包括指定目录中所有子目录(多层)，应该如何解决？

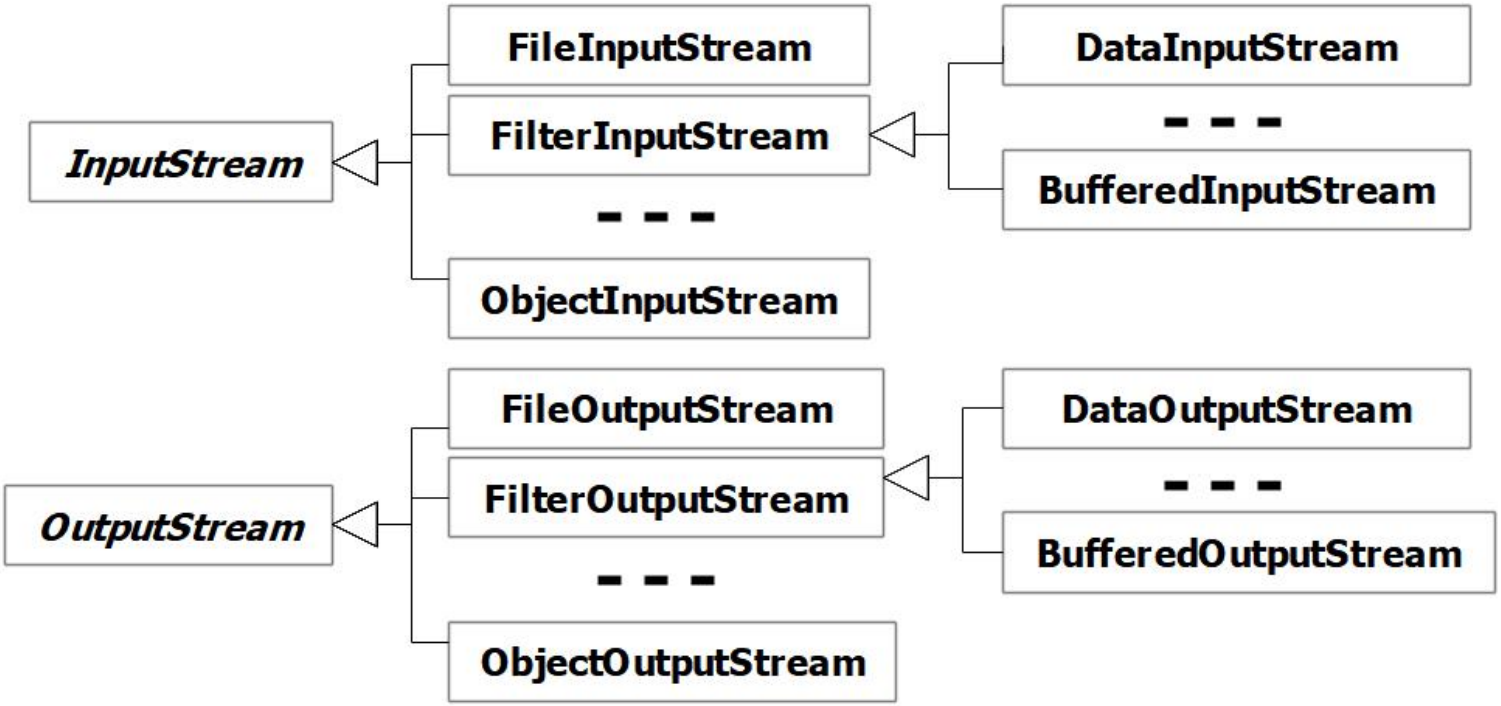
2 字节流与字符流



- 以**字节**为传输单位而创建的流称为字节流。
- 以**字符**为传输单位而创建的流称为字符流。

2.1 InputStream & OutputStream

- 抽象类InputStream是所有字节输入流的父类
- 抽象类OutputStream是所有字节输出流的父类



- 流操作的方法都声明抛出java.io.IOException或其后代类.

2.1 InputStream & OutputStream

InputStream中声明的方法

- int read()
- int read(byte[] b)
- int read(byte[] b, int off, int len)
- int available()
- void close()
- long skip(long n)
- boolean markSupported()
- void mark(int readlimit)
- void reset()

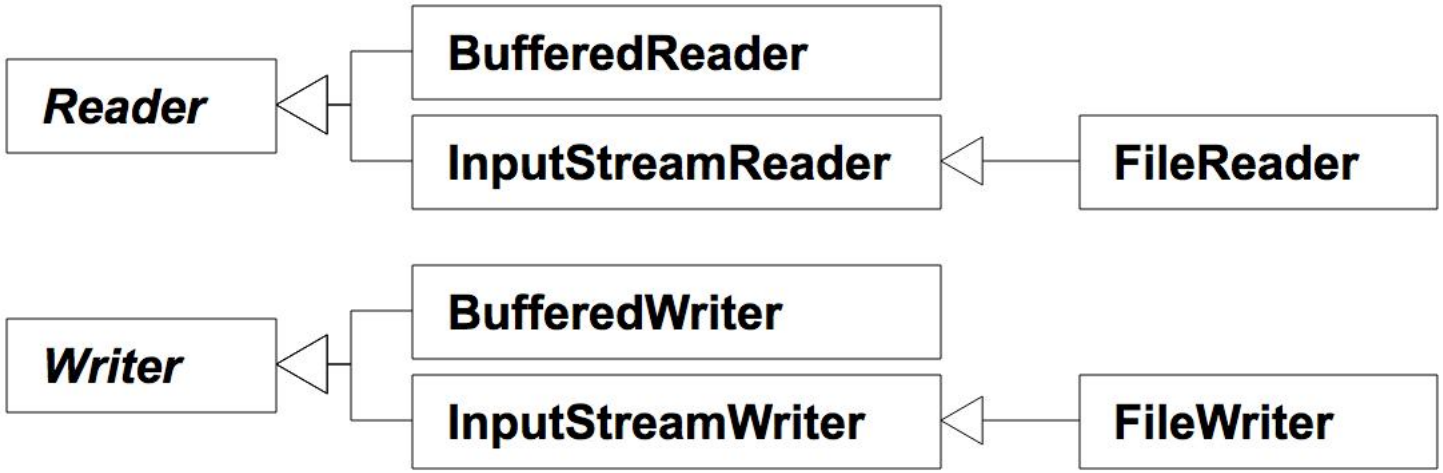
2.1 InputStream & OutputStream

OutputStream中声明的方法

- void write(int b)
- void write(byte[] b)
- void write(byte[] b, int off, int len)
- void close()
- void flush()

2.2 Reader & Writer

- 抽象类Reader是所有字符输入流的父类
- 抽象类Writer是所有字节输出流的父类



2.2 Reader & Writer

Reader中声明的方法

- int read()
- int read(char[] b)
- int read(char[] b, int off, int len)
- int available()
- void close()
- long skip(long n)
- boolean markSupported()
- void mark(int readlimit)
- void reset()

2.2 Reader & Writer

Writer中声明的方法

- void write(int b)
- void write(char[] b)
- void write(char[] b, int off, int len)
- void close()
- void flush()

3 文件字节流

A **FileInputStream** obtains input bytes from a file in a file system. 常用构造方法如下：

```
public FileInputStream(File file) throws FileNotFoundException
public FileInputStream(String name) throws FileNotFoundException
```

A **FileOutputStream** is an output stream for writing data to a File. 常用构造方法如下：

```
public FileOutputStream(File file) throws FileNotFoundException
public FileOutputStream(File file, boolean append)
                                    throws FileNotFoundException
public FileOutputStream(String name) throws FileNotFoundException
public FileOutputStream(String name, boolean append)
                                    throws FileNotFoundException
```

3 文件字节流

示例：使用文件字节流实现文件的复制.

```
public static void copy(File src, File tar) {
    FileInputStream srcStream = null;
    FileOutputStream desStream = null;
    try {
        srcStream = new FileInputStream(src);
        desStream = new FileOutputStream(tar);
        int data;
        while((data = srcStream.read())!=-1) {
            desStream.write(data);
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if(desStream!=null) {
            try {
                desStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if(srcStream!=null) {
           try {
                srcStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

传统关闭方式

```
public static void copyAutoClose(File src, File tar) {
    try(FileInputStream srcStream = new FileInputStream(src);
        FileOutputStream desStream = new FileOutputStream(tar)) {
        int data;
        while((data = srcStream.read())!=-1) {
           desStream.write(data);
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

自动关闭方式

完整代码：examples/Lecture08/FileCopyDemo

4 文件字符流

FileReader: Convenience class for reading character files.

```
public FileReader(String name) throws FileNotFoundException
public FileReader(File file) throws FileNotFoundException
```

FileWriter: Convenience class for writing character files.

```
public FileWriter(File file) throws IOException
public FileWriter(File file, boolean append) throws IOException
public FileWriter(String name) throws IOException
public FileWriter(String name, boolean append) throws IOException
```

5 缓冲流

以BufferedReader和BufferedWriter为例。

```
public BufferedReader(Reader in)
public BufferedWriter(Writer out)
```

缓冲流是高级流，高级流对象必须建立在字节流或字符流之上。

```
BufferedReader in
    = new BufferedReader(new FileReader("foo.in"));

PrintWriter out =
    new PrintWriter(
            new BufferedWriter(
                    new FileWriter("foo.out")
            )
     );
```

5 缓冲流

示例，使用BufferedReader和FileReader统计一个Java源程序文件如下数据：

(1) 文件的总行数

(2) 某个指定标识符的出现次数

```
public void count(File file, String word) {
    this.numberOfLines = 0;
    this.numberOfWord = 0;
    Pattern pattern = Pattern.compile("[\\W]" + word + "[\\W]");
    try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
        String line = null;
        // readLine()方法返回null，表示读取结束
        while ((line = reader.readLine()) != null) {
            this.numberOfLines++;
            Matcher matcher = pattern.matcher(" " + line);
            while (matcher.find()) {
                this.numberOfWord++;
            }
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

完整代码: examples/Lecture08/CodeLineCounter

6 对象流

An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.

ObjectOutputStream的基本用法示例：

```
FileOutputStream fos = new FileOutputStream("t.tmp");
ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeInt(12345);                //写基本类型 writeXXX
oos.writeUTF("Today");              //写字符串的方法
oos.writeObject(new Date()); //写对象的方法

oos.close();
```

6 对象流

An ObjectInputStream deserializes primitive data and objects previously written using an ObjectOutputStream.

ObjectInputStream的基本用法示例：

```
FileInputStream fis = new FileInputStream("t.tmp");
ObjectInputStream ois = new ObjectInputStream(fis);

int i = ois.readInt();
String today = ois.readUTF();
Date date = (Date) ois.readObject();

ois.close();
```

完整示例代码：examples/Lecture08/object_stream_demo

7 对象序列化

- 一个类实现了 java.io.Serializable 接口，则该类的对象就是可序列化对象。
- 使用 ObjectOutputStream 和 ObjectInputStream 能够对可序列化对象进行读写操作。
- java.io.Serializable 是标记型接口，实现时无需重写方法。

例如：

```
public class Student implements java.io.Serializable {
    //.....
}
```

完整示例代码：examples/Lecture08/object_stream_demo

7 对象序列化

- 一个类实现了 java.io.Serializable 接口，该类的对象组成的数组也是可序列化对象
- ArrayList、HashSet等常用数组结构类也实现了 java.io.Serializable 接口，因此如果其中的元素都是可序列化对象，则该数组结构对象也是可序列化对象。

例如：

```
public class Student implements java.io.Serializable {
    //.....
}
Student[] students = new Student[10];
ArrayList<Student> list = new ArrayList<>();
```

上面的数组students和线性表都是可序列化的。

完整示例代码：examples/Lecture08/object_stream_demo

课后工作

- 结合教材和Java API文档，要求掌握：
  - File类的使用
  - InputStream和OutputStream的基本原理
  - 使用FileInputStream和FileOutputStream以字节为单位对文件进行读写
  - 使用ObjectInputStream和ObjectOutputStream以对象为单位对文件进行读写
  - 对象序列化的方法
  - Reader和Writer的基本原理
  - 使用BufferedReader、BufferedWriter结合FileReader和FileWriter对文本文件进行读写
- 自行练习教材的上机实践和习题