

JAVA程序设计



第8章 I/O流

《Java程序设计》



第1章 Java语言基础 (2)

第2章 结构化程序设计基础 (3)

第3章 Java类与对象 (3)

第4章 Java类的继承与多态 (6)

第5章 图形化用户界面 (4)

第6章 异常处理 (2)

第7章 对象的容纳 (4)

第8章 IO流 (2)

第9章 并发控制 (2)

第10-11章 高级程序设计 (4)



本章目录

- ◆ 8.1 文件的读取I/O
- ◆ 8.2 流与相关类
- ◆ 8.3 标准I / O流
- ◆ 8.4 随机访问文件

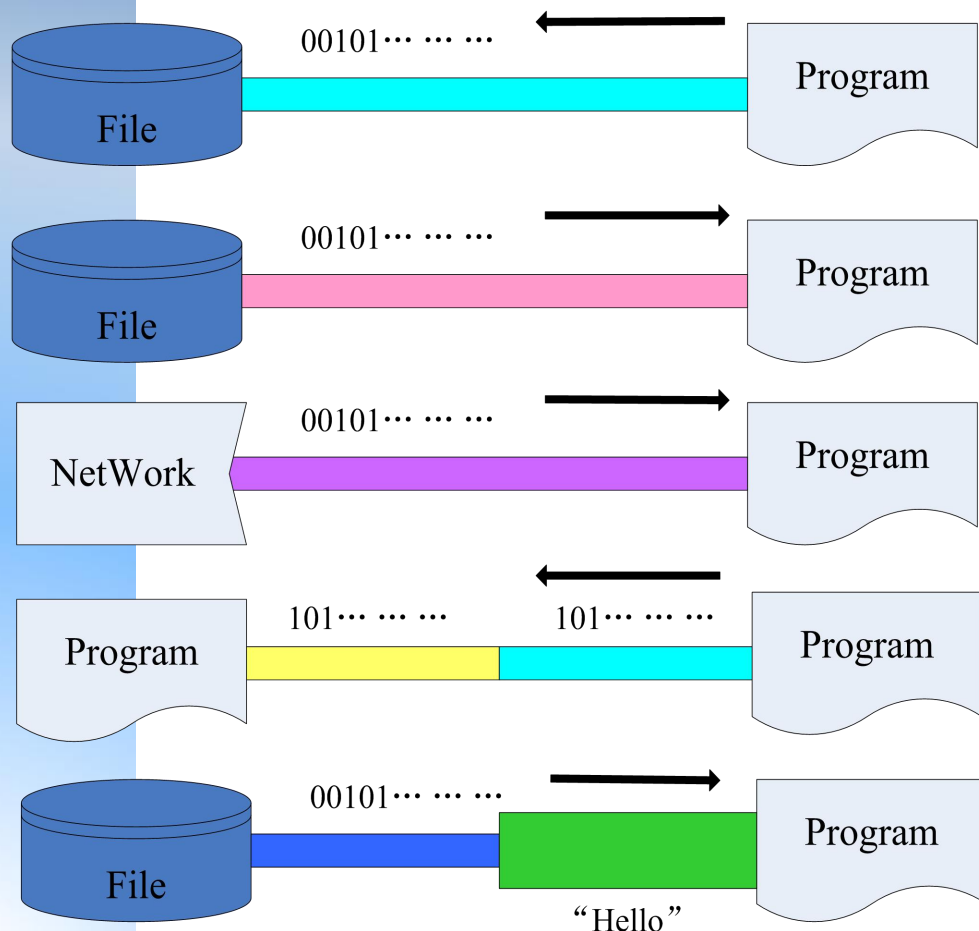


I/O介绍

➤ 创建一个好的输入/输出系统是一项艰难的任务。

- 与不同的源和目的端进行交互，包括文件、控制台、网络链接等；
- 以不同的方式与它们进行通信（顺序、随机存取、缓冲、二进制、按照字符、按行、按字节等）；
- 大多数I/O需要进行异常处理。

1. 流的概念



➤流是Java语言输入/输出的方式，Java语言程序通过流来完成输入/输出工作。

➤Java提供了各种各样的流类，来实现IO，封装了数据处理的细节。



8.1 文件的读取

文件

一个文件包含 n Bytes

0 1 2 3 4 $n-1$ -1

Sally	Black		
Tom	Blue		
Judy	Green		
Iris	Orange		
Randy	Red		

Judy Green

line

Object Serializable

J u d y Character(String)

Data double/boolean/...

01001010 Byte

1 Bit



8.1 文件的读取

File类

java.io.File

➤ 文件和目录路径名的抽象表示形式，用来获取文件或目录的信息。

File类的构造器：

➤ **public File (String pathname)**

//以pathname为路径创建File对象，如果pathname为相对路径则在默认的当前路径下进行存储

➤ **public File(String parent, String child)**

//以parent为父路径，child为子路径创建File对象



8.1 文件的读取

File类的方法:

- **public boolean canRead()**
- **public boolean canWrite()**
- **public boolean exists()**
- **public boolean isDirectory()**
- **public boolean isFile()**
- **public boolean isHidden()**
- **public long lastModified()**
- **public long length()**
- **public String getName()**
- **public String getPath()**
- **public boolean createNewFile()**
- **public boolean delete()**
- **public boolean mkdir()**
- **public boolean mkdirs ()**



8.1 文件的读取

```
import java.io.*;
public class Test {
    public static void main (String[] args) {
        String filename = "myfile.txt";
        String directory = "mydir1" + "/" + "mydir2";
        File f = new File(directory, filename);
        if (f.exists()) {
            System.out.println("File Name: " + f.getAbsolutePath());
            System.out.println("File Length: " + f.length());
        } else {
            f.getParentFile().mkdirs();
            try {
                f.createNewFile() ;
            } catch (IOException e) {
                e.printStackTrace() ;
            }
        }
    }
}
```

```
File Name: D:\workspace\test2\mydir1\mydir2\myfile.txt
File Length: 0
```



8.1 文件的读取

File类的方法:

- **public boolean canRead()**
- **public boolean canWrite()**
- **public boolean exists()**
- **public boolean isDirectory()**
- **public boolean isFile()**
- **public boolean isHidden()**
- **public long lastModified()**
- **public long length()**
- **public String getName()**
- **public String getPath()**
- **public boolean createNewFile()**
- **public boolean delete()**
- **public boolean mkdir()**
- **public boolean mkdirs ()**

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    File file = new File();  
    file.  
}
```

A screenshot of an IntelliJ IDEA autocomplete popup window. The window is titled 'list()' and shows a list of methods from the File class. The methods are: lastModified() : long - File, length() : long - File, list() : String[] - File (highlighted), list(FilenameFilter filter) : String[] - File, listFiles() : File[] - File, listFiles(FileFilter filter) : File[] - File, listFiles(FilenameFilter filter) : File[] - File, and listRoots() : File[] - File. At the bottom of the popup, it says 'Press 'Alt+/' to show Template Proposals'.



8.1 文件的读取

- 读文件

示例：用**FileReader**读入一个文件并显示在屏幕上。

import java.io.*;

```
import java.io.*;

public class TestFileReader {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        //加入中文
        FileReader fr = null;
        int c = 0;
        try {
            fr = new FileReader("./src/TestFileReader.java");
            int ln = 0;
            while ((c = fr.read()) != -1) {
                System.out.print((char) c);
            }
            fr.close();
        } catch (FileNotFoundException e) {
        } catch (IOException e) {
        }
    }
}
```

catch (IOException e) { }



8.1 文件的读取

- 写文件

示例：用**FileWriter**写一个包含字母**A-Z**的文件。

```
import java.io.*;
public class TestFileWriter {
    public static void main(String[] args) {
        FileWriter fw = null;
        try {
            fw = new FileWriter("./unicode.dat");
            for(char c='A';c<='Z';c++){
                fw.write(c); }
                fw.close();
        } catch (IOException e1) {
            e1.printStackTrace();
            System.out.println("File write error");
            System.exit(-1); }}}}
```



8.1 文件的读取

- 文件存储格式

文件

一个文件包含 n Bytes

0 1 2 3 4 $n-1$ -1

Sally	Black		
Tom	Blue		
Judy	Green		
Iris	Orange		
Randy	Red		

Judy Green

line

Object Serializable

Data double/boolean/...

J u d y Character(String)

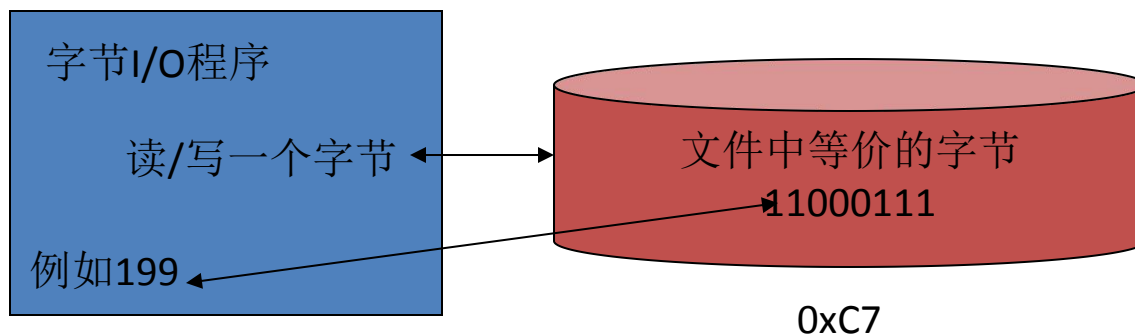
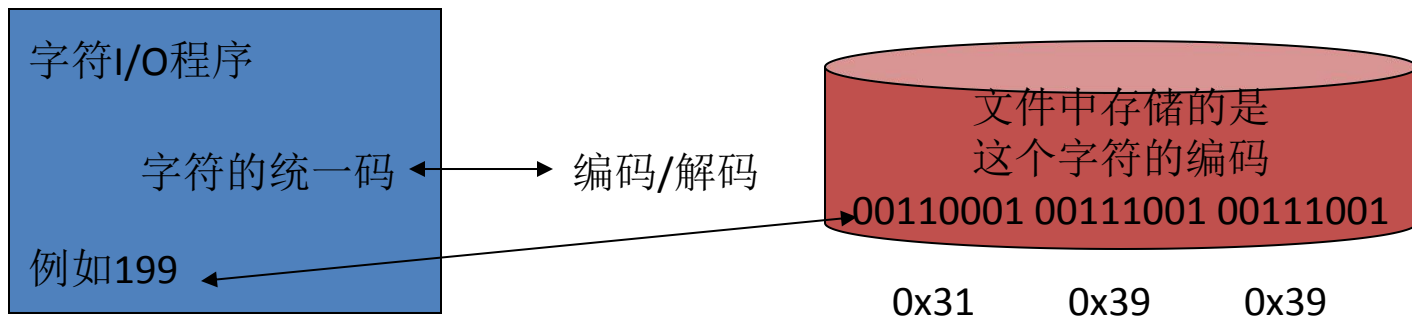
01001010 Byte

1 Bit



8.1 文件的读取

- 文件存储格式





8.1 文件的读取

- 字节文件读（二进制）

示例：用**FileInputStream**读入一个文件并显示在屏幕上。

```
import java.io.*;

public class TestFileInputStream {
    public static void main(String[] args) {
        int b = 0;
        FileInputStream in = null;
        try {
            in = new
                FileInputStream("./TestFileReader.java");
        } catch (FileNotFoundException e) {
            System.out.println("Can't find file");
            System.exit(-1);
        }
    }
}
```



8.1 文件的读取

- 字节文件读（二进制）

```
try {  
    long num = 0;  
    while((b=in.read())!=-1){  
        System.out.print((char)b);  
        num++;  
    }  
    in.close();  
    System.out.println();  
    System.out.println("read "+num+" Byte");  
} catch (IOException e1) {  
    System.out.println("File read error");  
    System.exit(-1);  
}}}
```

.....（显示源程序码）

read 768 Byte

```
import java.io.*;
```

```
public class TestFileReader {
```

```
    public static void main(String[] args) {
```

```
        // TODO Auto-generated method stub
```

```
        //?????????
```

```
        FileReader fr = null;
```

```
        int c = 0;
```

```
        try {
```

```
            fr = new FileReader("./src/TestFileReader.java");
```

```
            int ln = 0;
```

```
            while ((c = fr.read()) != -1) {
```

```
                System.out.print((char) c);
```

```
            }
```

```
        import java.io.*;
```

```
    } ca
```

```
    } ca
```

```
    } public class TestFileReader {
```

```
        }
```

```
}
```

read 436 Byte

```
        public static void main(String[] args) {
```

```
            // TODO Auto-generated method stub
```

```
            //加入中文
```

```
            FileReader fr = null;
```

```
            int c = 0;
```

```
            try {
```

```
                fr = new FileReader("./src/TestFileReader.java");
```

```
                int ln = 0;
```

```
                while ((c = fr.read()) != -1) {
```

```
                    System.out.print((char) c);
```

```
                }
```

```
                fr.close();
```

```
            } catch (FileNotFoundException e) {
```

```
            } catch (IOException e) {
```

```
            }
```

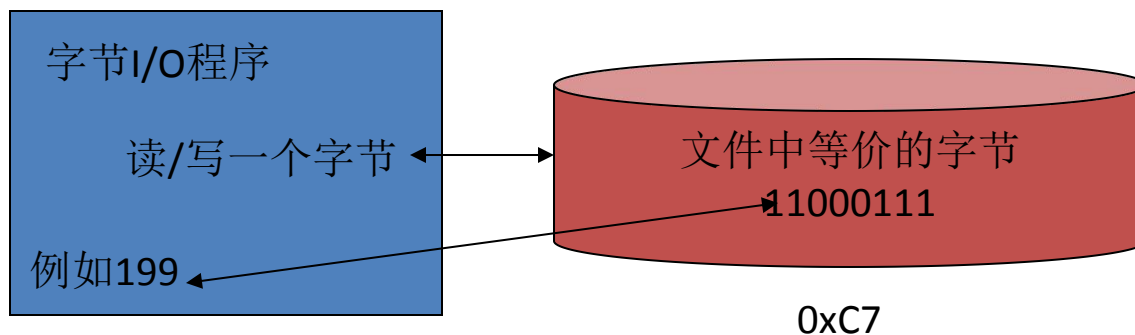
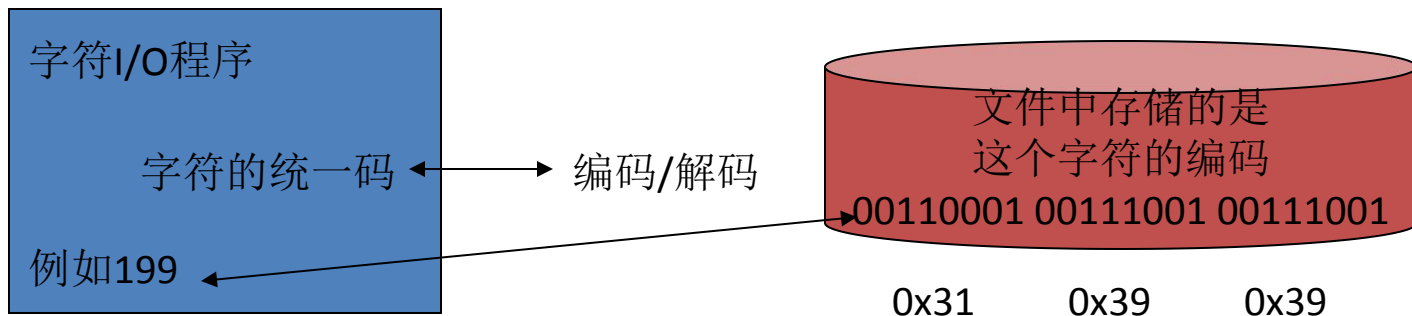
```
        }
```

```
    }
```



8.1 文件的读取

- 文件存储格式





8.1 文件的读取

- 字节文件写（二进制）

示例：用**FileInputStream**和**FileOutputStream**实现文件的复制。

```
import java.io.*;
public class TestFileOutputStream {
    public static void main(String[] args) {
        int b = 0;
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("./HelloWorld.java");
            out = new FileOutputStream("./HW.java");
            while((b=in.read())!=-1){
                out.write(b);
            }
        }
```



8.1 文件的读取

```
in.close();  
out.close();  
} catch (FileNotFoundException e2) {  
    System.out.println("Can't find file");  
    System.exit(-1);  
} catch (IOException e1) {  
    System.out.println("File copy error");  
    System.exit(-1);  
}  
System.out.println("File copy finished");  
}  
}
```

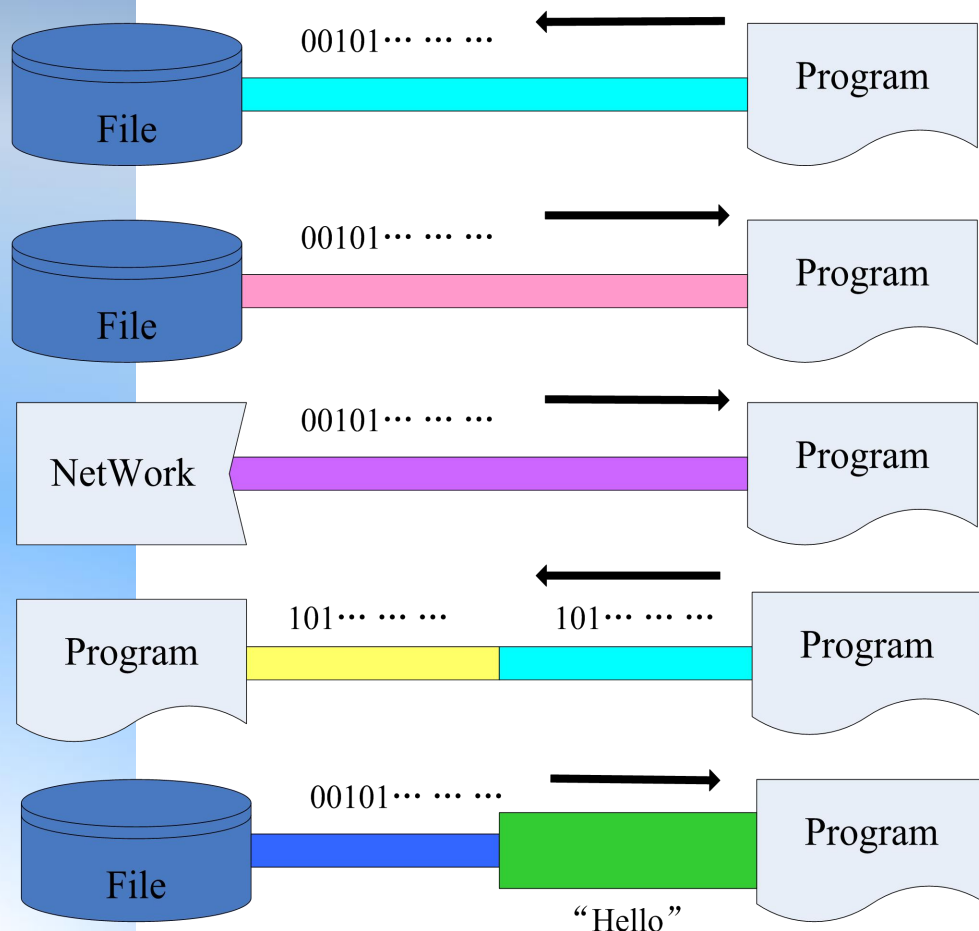
File copy finished



8.1 文件的读取

- 字符文件
 - `FileReader`
 - `FileWriter`
- 字节文件
 - `FileInputStream`
 - `FileOutputStream`

1. 流的概念



➤流是Java语言输入/输出的方式，Java语言程序通过流来完成输入/输出工作。

➤Java提供了各种各样的流类，来实现IO，封装了数据处理的细节。



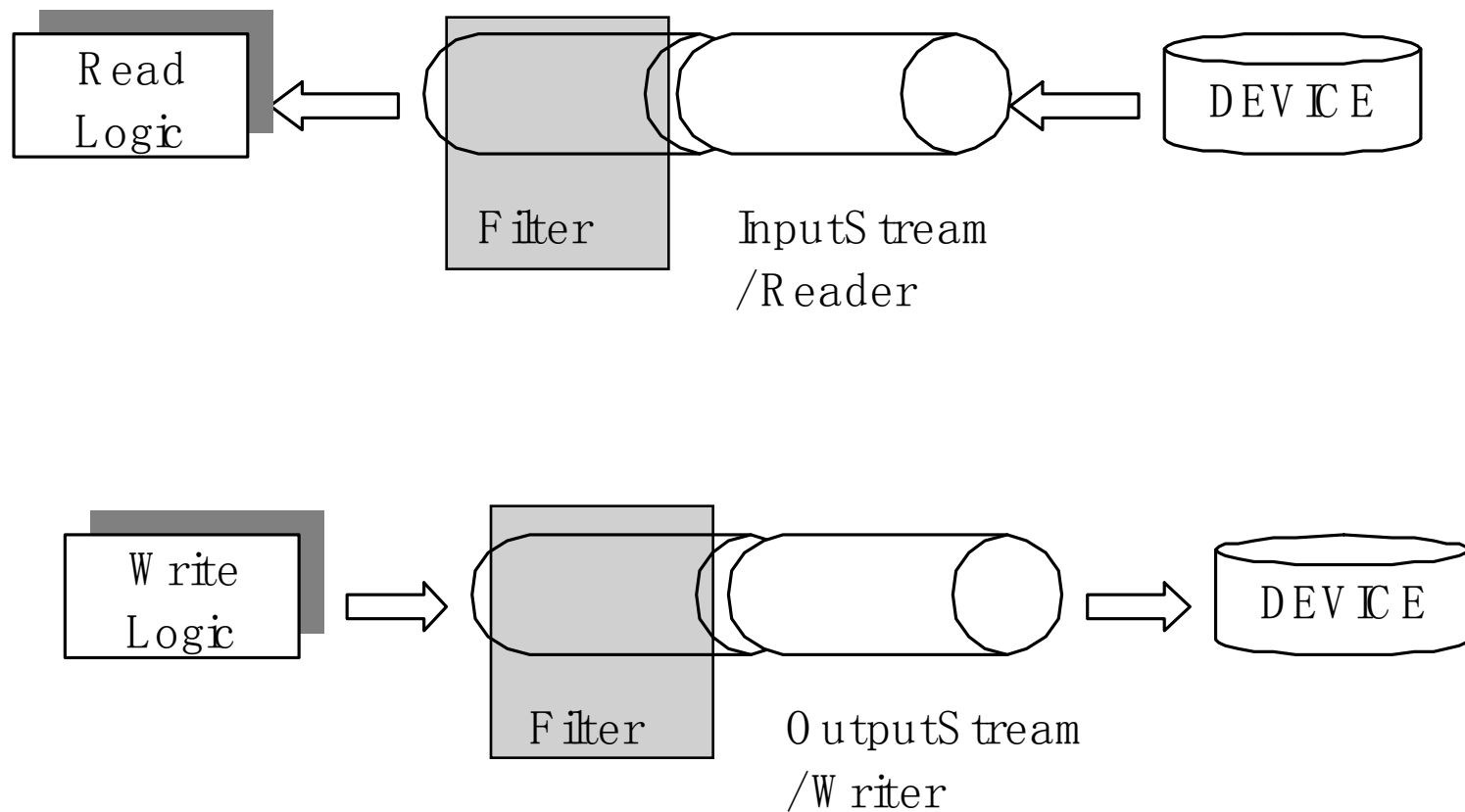
8.2 流与相关类

流（stream）是一个想象中的无限长的数据序列。

- 按数据流向分为：输入流、输出流
- 按处理数据单位分为：字节流、字符流
- 按功能分为：节点流、处理流

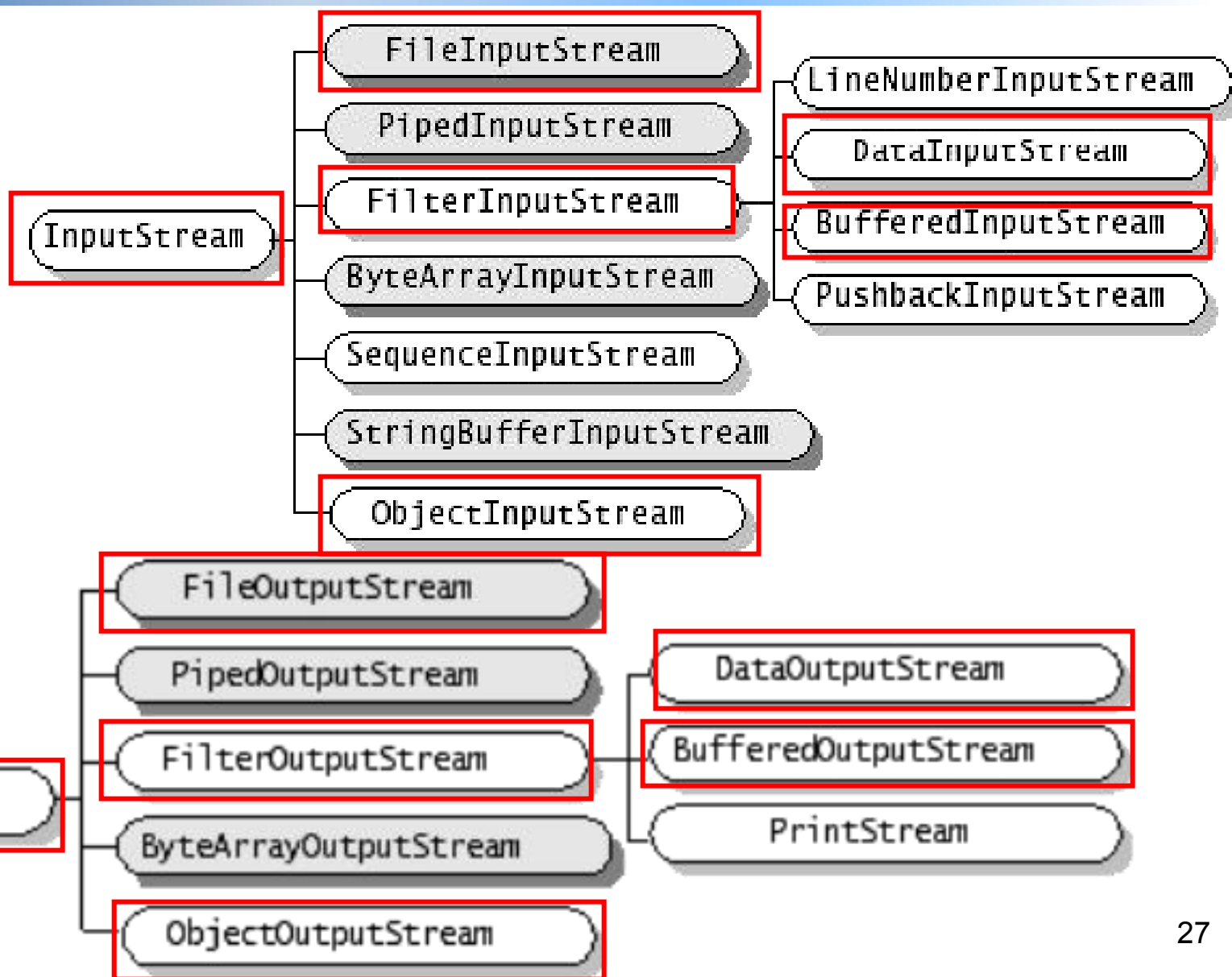
	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

1. 流的概念





字节流





字节流

InputStream的方法 (throws IOException)

➤ int read()

- 读取一个字节并以整数的形式返回

➤ int read(byte[] buffer)

- 读取一系列字节并存储到一个数组buffer中，返回读取的字节数

➤ int read(byte[] buffer, int offset, int length)

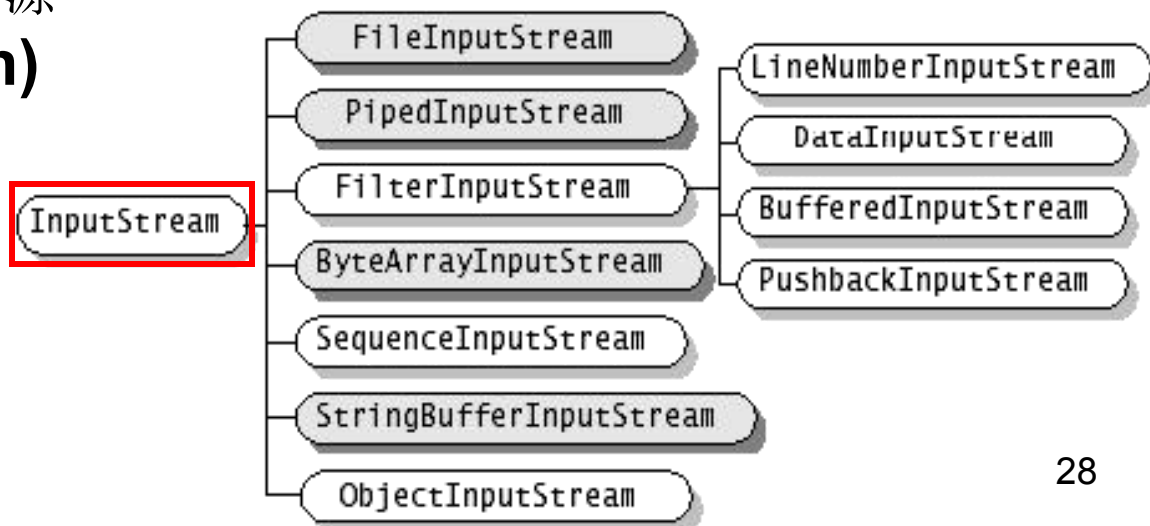
- 从offset位置存储到一个数组buffer中

➤ void close()

- 关闭释放内存资源

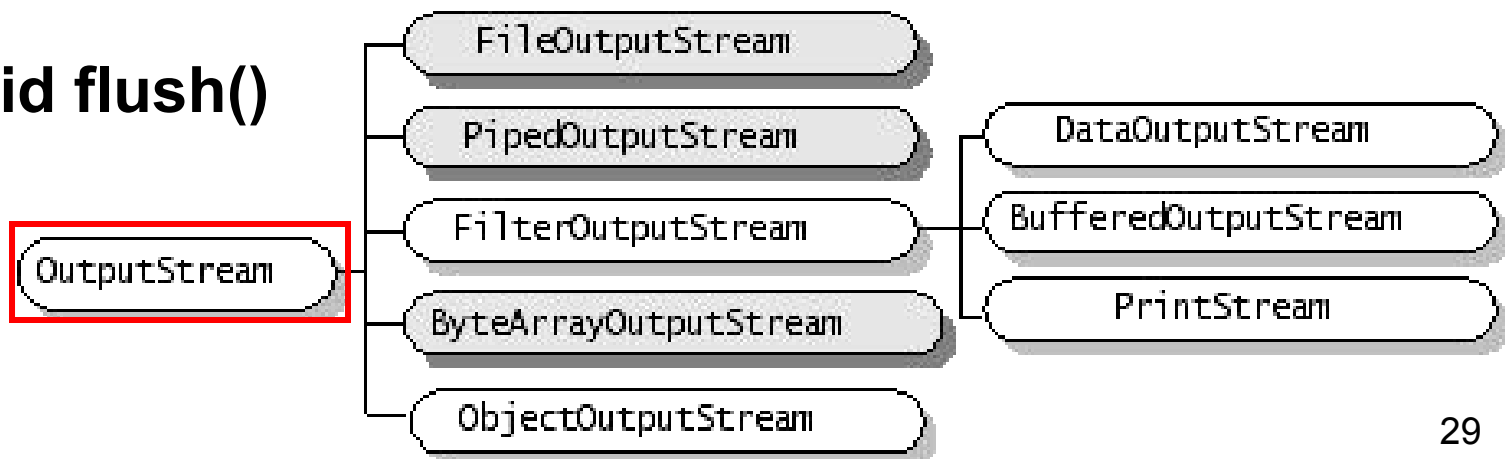
➤ long skip(long n)

- 跳过n个字节



OutputStream的方法 (throws IOException)

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





字节流

InputStream的子类型

类	功能
ByteArray-InputStream	允许将内存的缓冲区当作 InputStream 使用
StringBuffer-InputStream	将 String 转换成 InputStream
File-InputStream	用于从文件中读取信息
Piped-InputStream	产生用于写入相关 PipedOutput-Stream 的数据。实现“管道化”概念。
Sequence-InputStream	将两个或多个 InputStream 对象转换成单一 InputStream 。



字节流

FilterInputStream的子类型

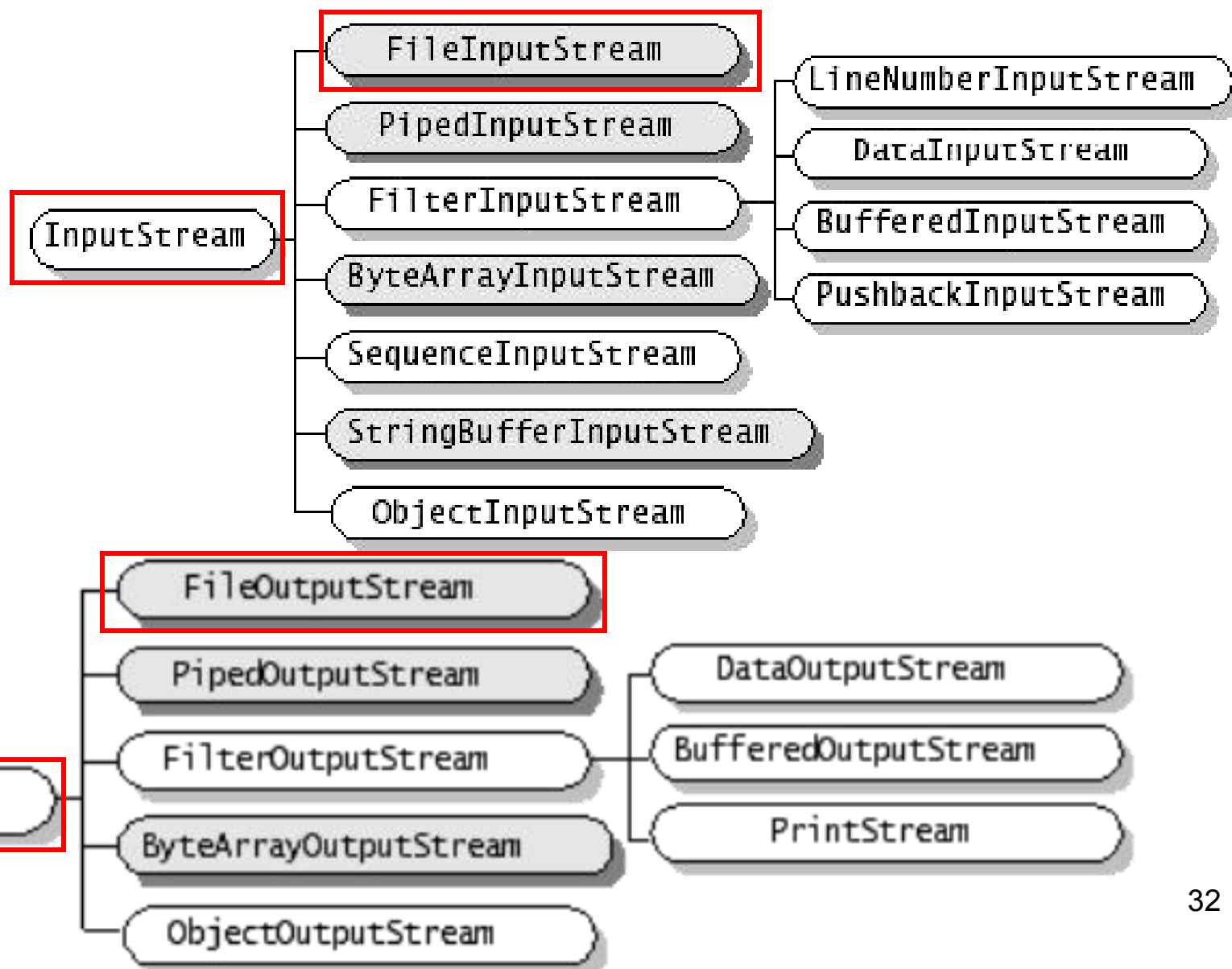
Data-InputStream	与 DataOutputStream 搭配使用，因此我们可以参照可移植方式从流读取基本数据类型 (int , char , long , etc.)
Buffered-InputStream	使用它可以防止每次读取时都得进行实际的读操作。代表“使用缓冲区”。
LineNumber-InputStream	跟踪输入流中的行号；可调用 getLineNumber() 和 setLineNumber(int) 。
Pushback-InputStream	具有一个字节的回退缓冲区，因此可以将读到的最后一个字符回退。

ObjectInputStream 将会在下一部分进行介绍。

OutputStream和**FilterOutputStream**的类型和**InputStream**、**FilterInputStream**类似



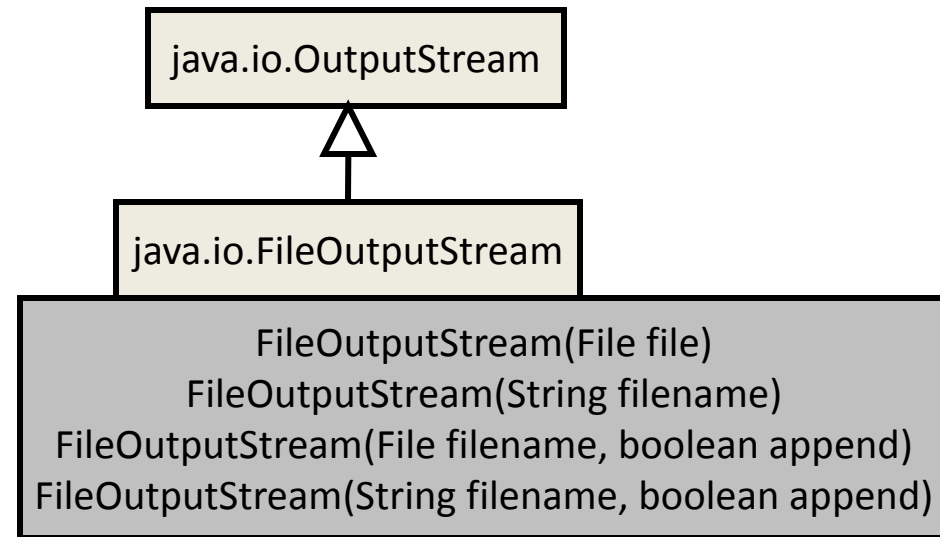
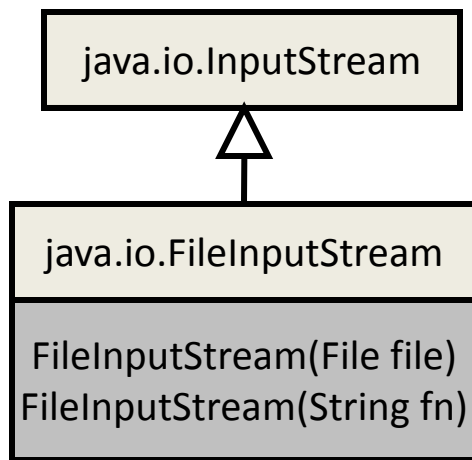
字节流





字节流

- FileInputStream类、FileOutputStream类
 - 所有方法都是从InputStream和OutputStream继承
- 构造方法





字节流

- FileInputStream类、FileOutputStream类
 - 为不存在文件创建FileInputStream对象，会发生java.io.FileNotFoundException异常。
 - FileOutputStream，如果文件存在，前两个构造方法删除文件当前内容，后两个方法当append为true时为文件追加内容。
 - 几乎所有的I/O类中的方法都会抛出异常java.io.IOException，必须在方法中声明或用try-catch



字节流

- FileInputStream类、FileOutputStream

```
import java.io.*;
public class TestFileStream {
    public static void main(String[] args) throws IOException{
        FileOutputStream output = new FileOutputStream("temp.txt");
        for(int i =1; i<10; i++)
            output.write(i);
        output.close();

        FileInputStream input = new FileInputStream("temp.txt");
        int value;
        while((value = input.read()) != -1)
            System.out.print(value + " ");
        input.close();
    }
}
```

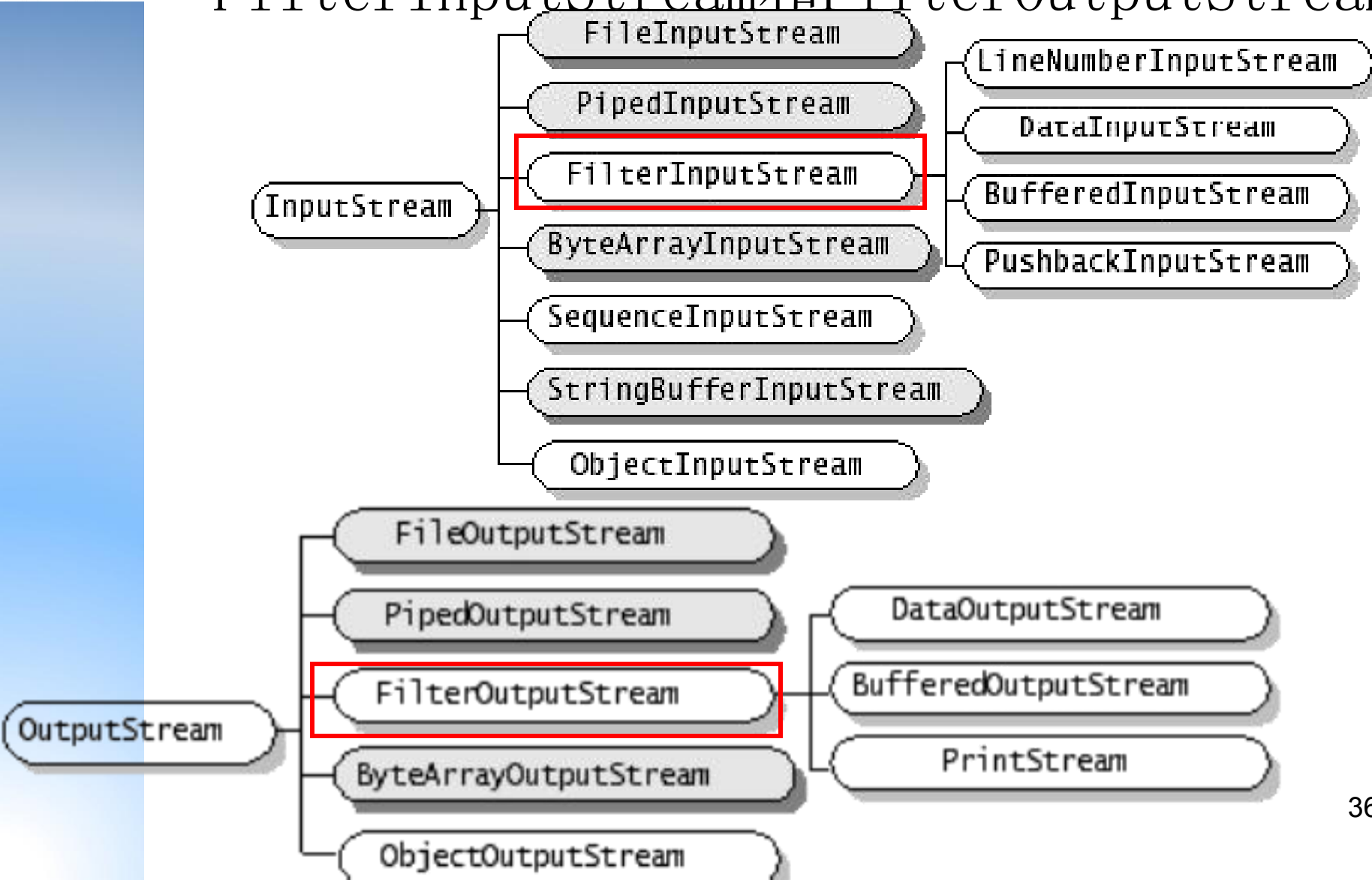
<terminated> TestFileStream [Ja

1 2 3 4 5 6 7 8 9



字节流

- FilterInputStream和FilterOutputStream

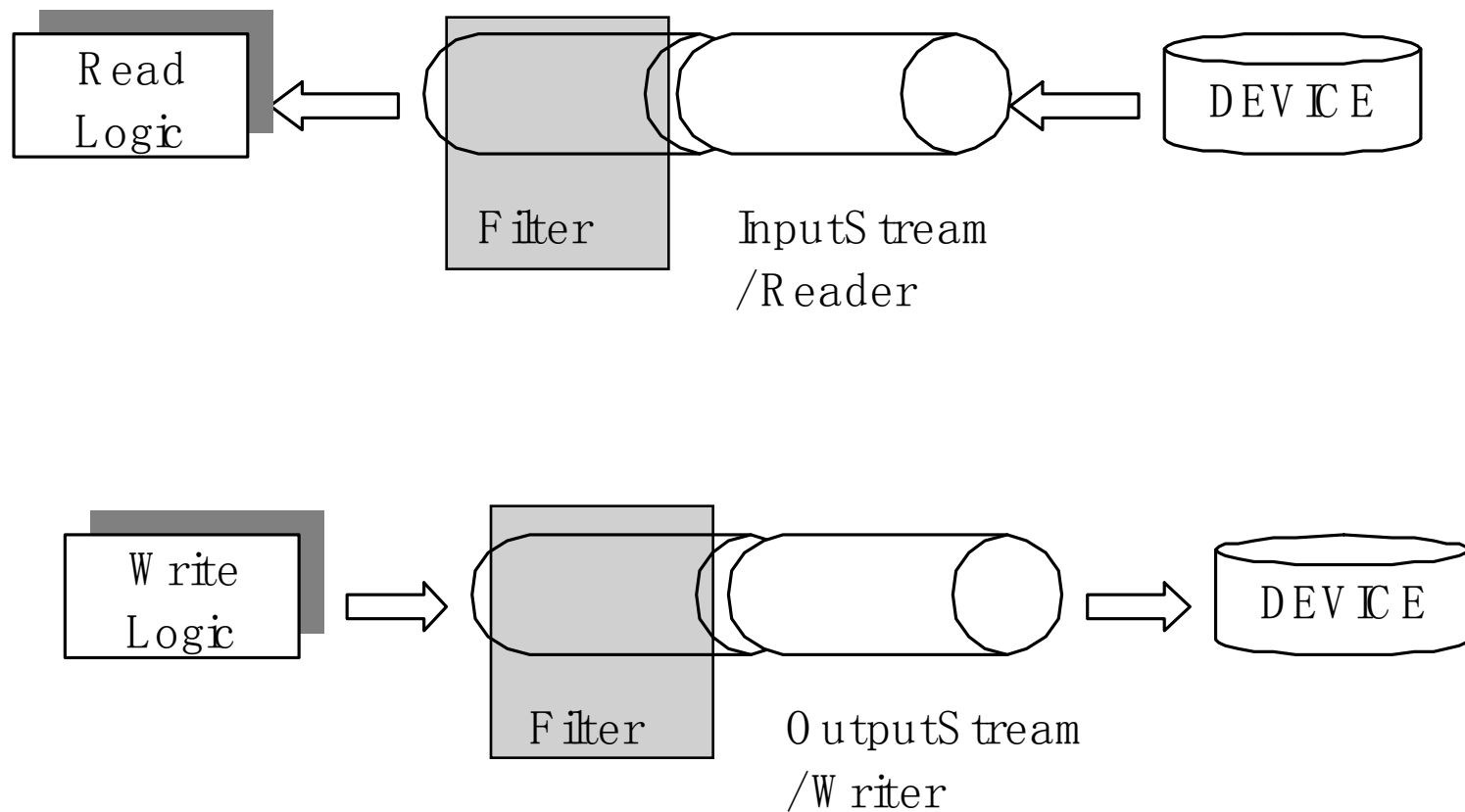




字节流

- `FilterInputStream`和`FilterOutputStream`
 - `FileInputStream`和`FileOutputStream`只能读取字节，如果要读写整数、双精度值或字符串，那么就需要过滤器类来包装输入流。
 - `FilterInputStream`和`FilterOutputStream`是过滤数据的基类，如果处理基本数值类型时，就使用`DataInputStream`和`DataOutputStream`类。

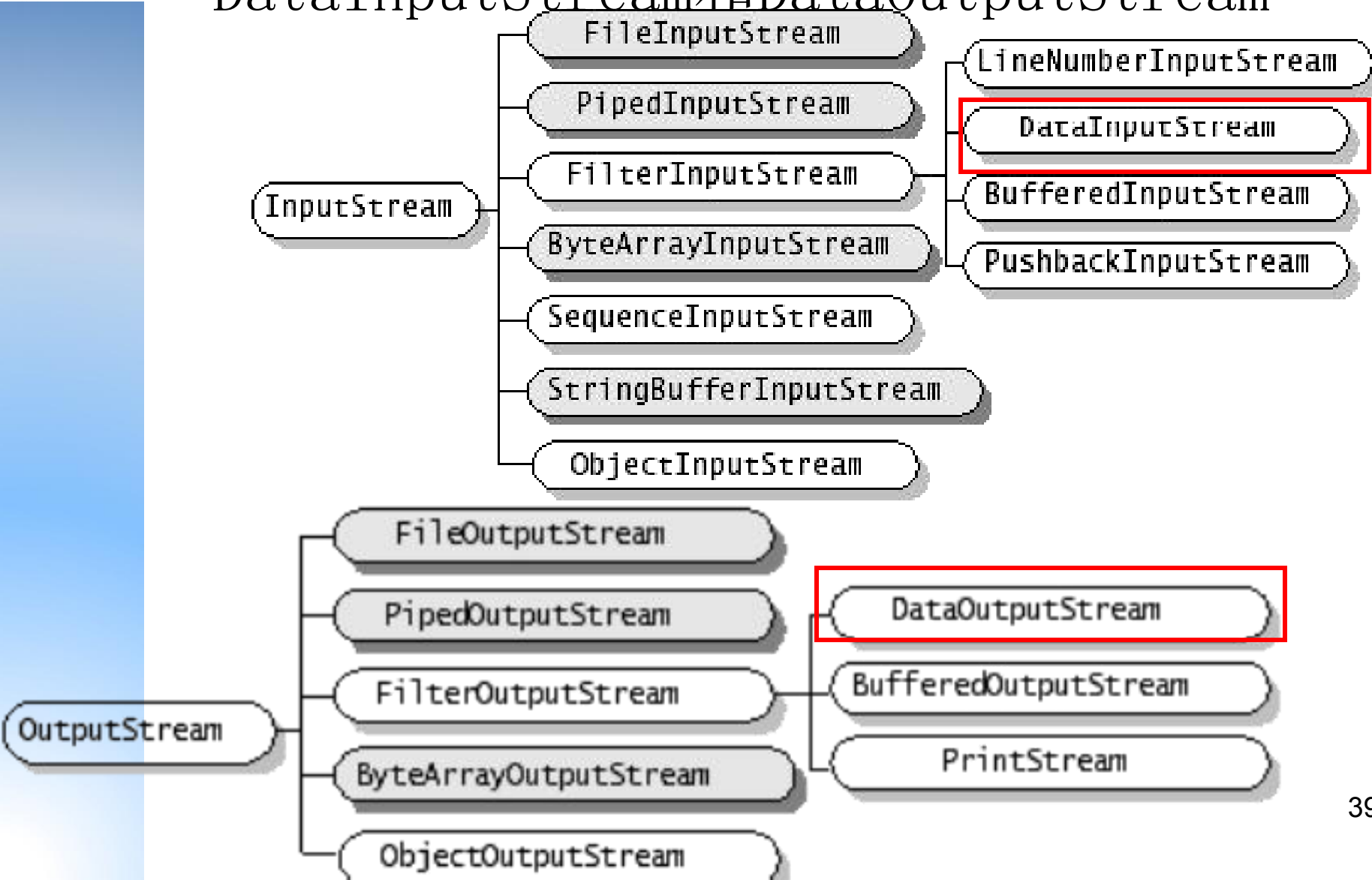
1. 流的概念





字节流

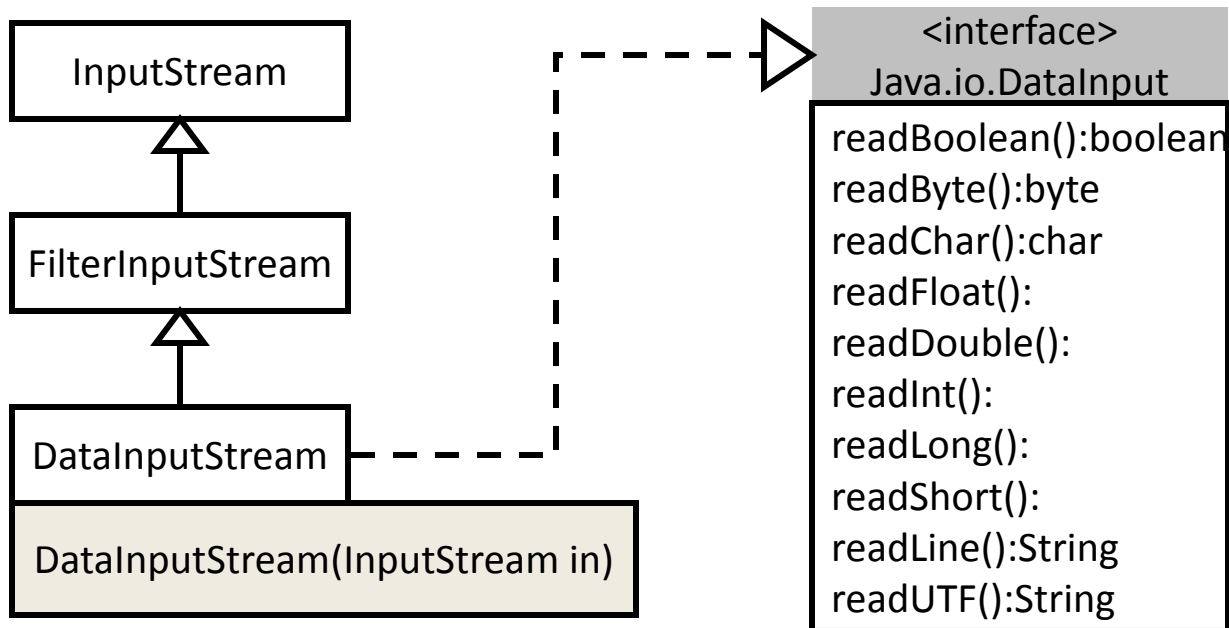
- DataInputStream和DataOutputStream





字节流

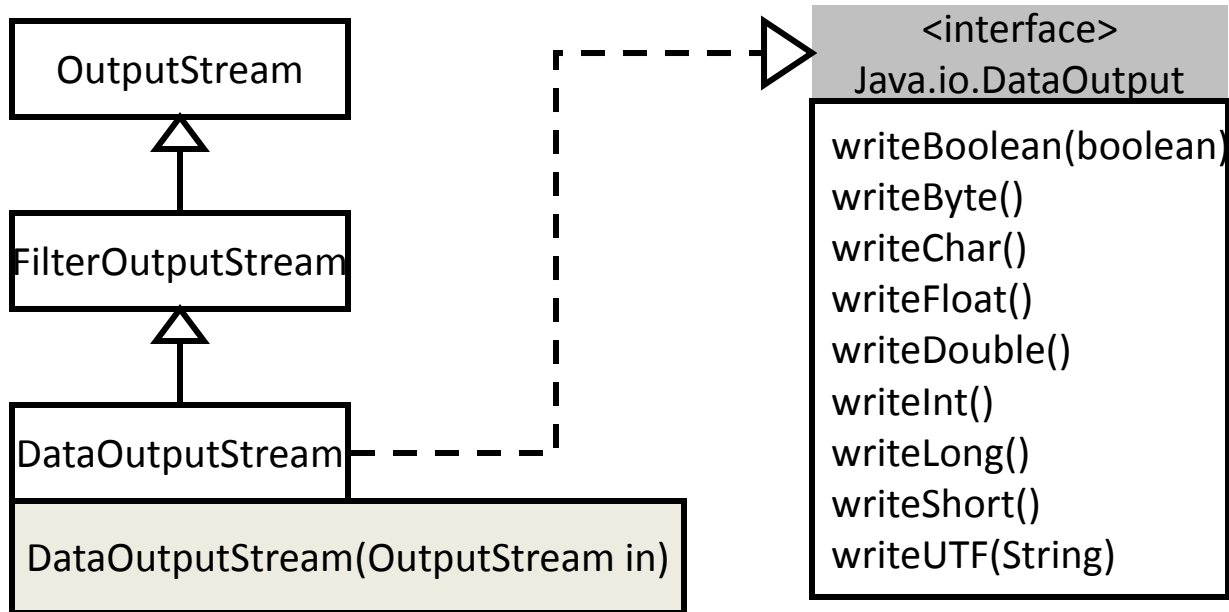
- DataInputStream和DataOutputStream





字节流

- DataInputStream和DataOutputStream





字节流

- DataInputStream和DataOutputStream
 - 用于对已经存在的输入/输出流进行**包装**，以便在原始流中过滤数据。
 - `public DataInputStream(InputStream instream)`
 - `public DataOutputStream(OutputStream outstream)`
 - `DataInputStream input =
 new DataInputStream(new
 FileInputStream(“in.dat”))`
 - `DataOutputStream output =
 new DataOutputStream(new
 FileOutputStream(“in.dat”))`



字节流

- DataInputStream和DataOutputStream

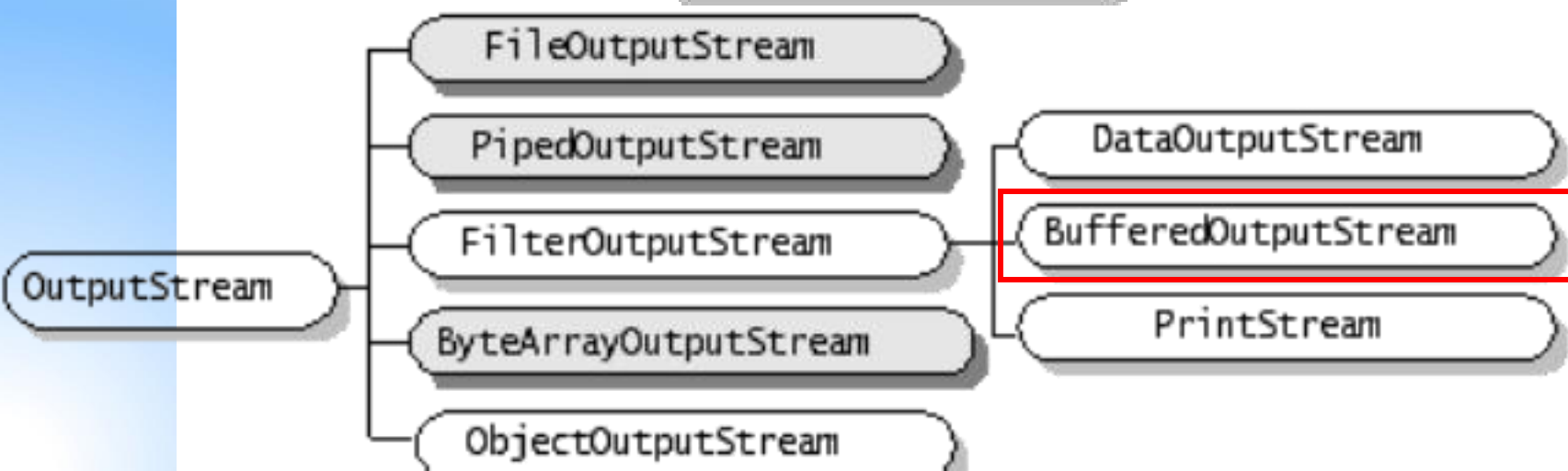
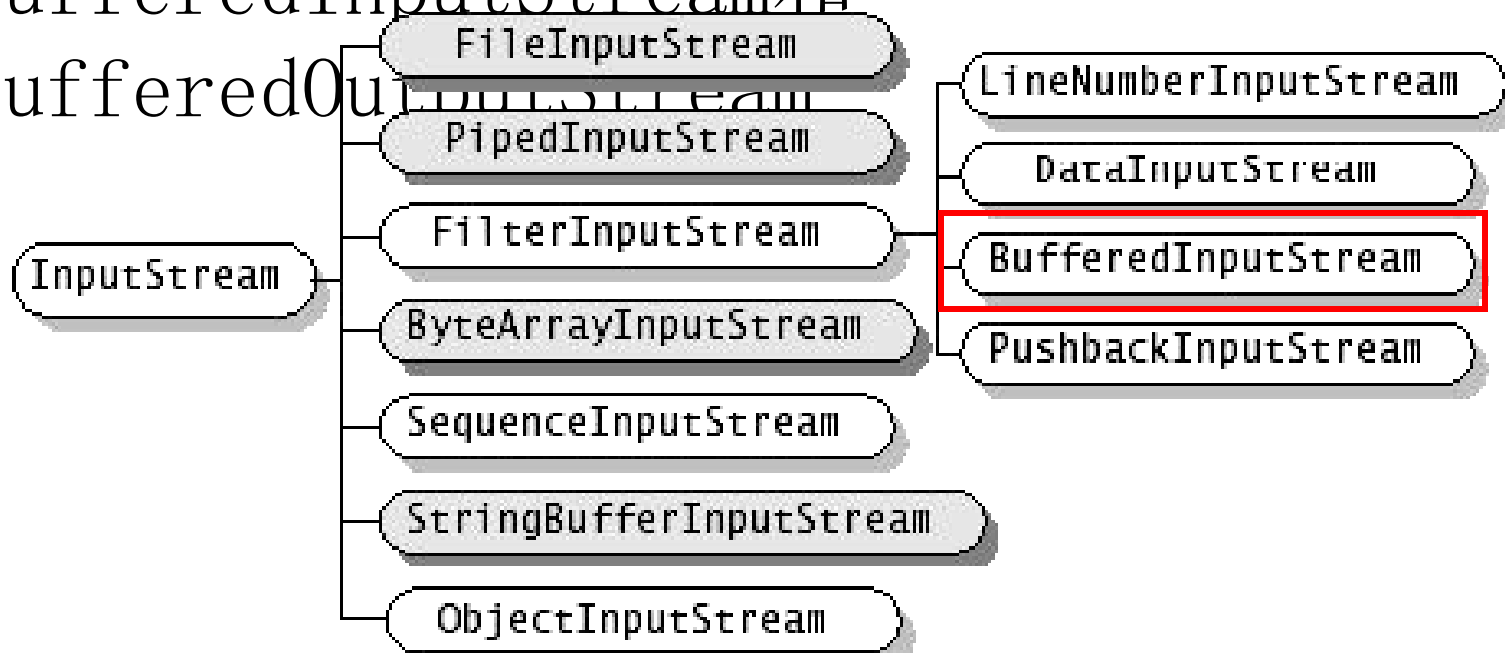
```
import java.io.*;
public class TestDataStream {
    public static void main(String[] args) throws IOException {
        DataOutputStream output = new DataOutputStream(new FileOutputStream("temp.dat"));
        output.writeUTF("John");
        output.writeDouble(85.5);
        output.writeUTF("Jim");
        output.writeDouble(90.5);
        output.writeUTF("George");
        output.writeDouble(80.7);
        output.close();

        DataInputStream input = new DataInputStream(new FileInputStream("temp.dat"));
        System.out.println(input.readUTF() + " " + input.readDouble());
        System.out.println(input.readUTF() + " " + input.readDouble());
        System.out.println(input.readUTF() + " " + input.readDouble());
        input.close();
    }
}
```

```
John 85.5
Jim 90.5
George 80.7
```

字节流

- BufferedInputStream和BufferedOutputStream





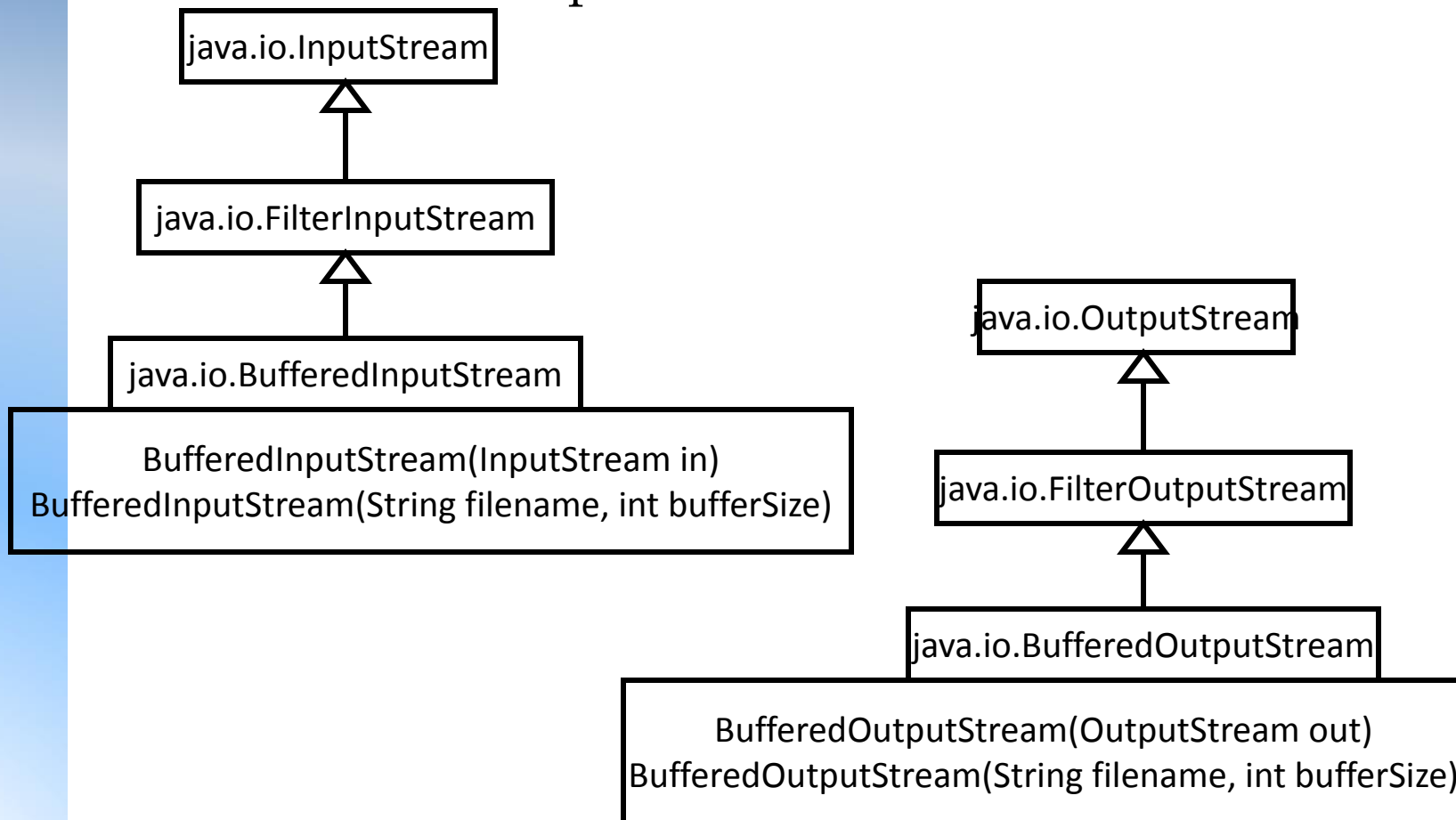
字节流

- BufferedInputStream和BufferedOutputStream
 - 减少读写次数（高速设备与低速设备之间）
 - 方法从InputStream和OutputStream继承而来
 - 没有新方法
 - 提高效率



字节流

- BufferedInputStream和BufferedOutputStream





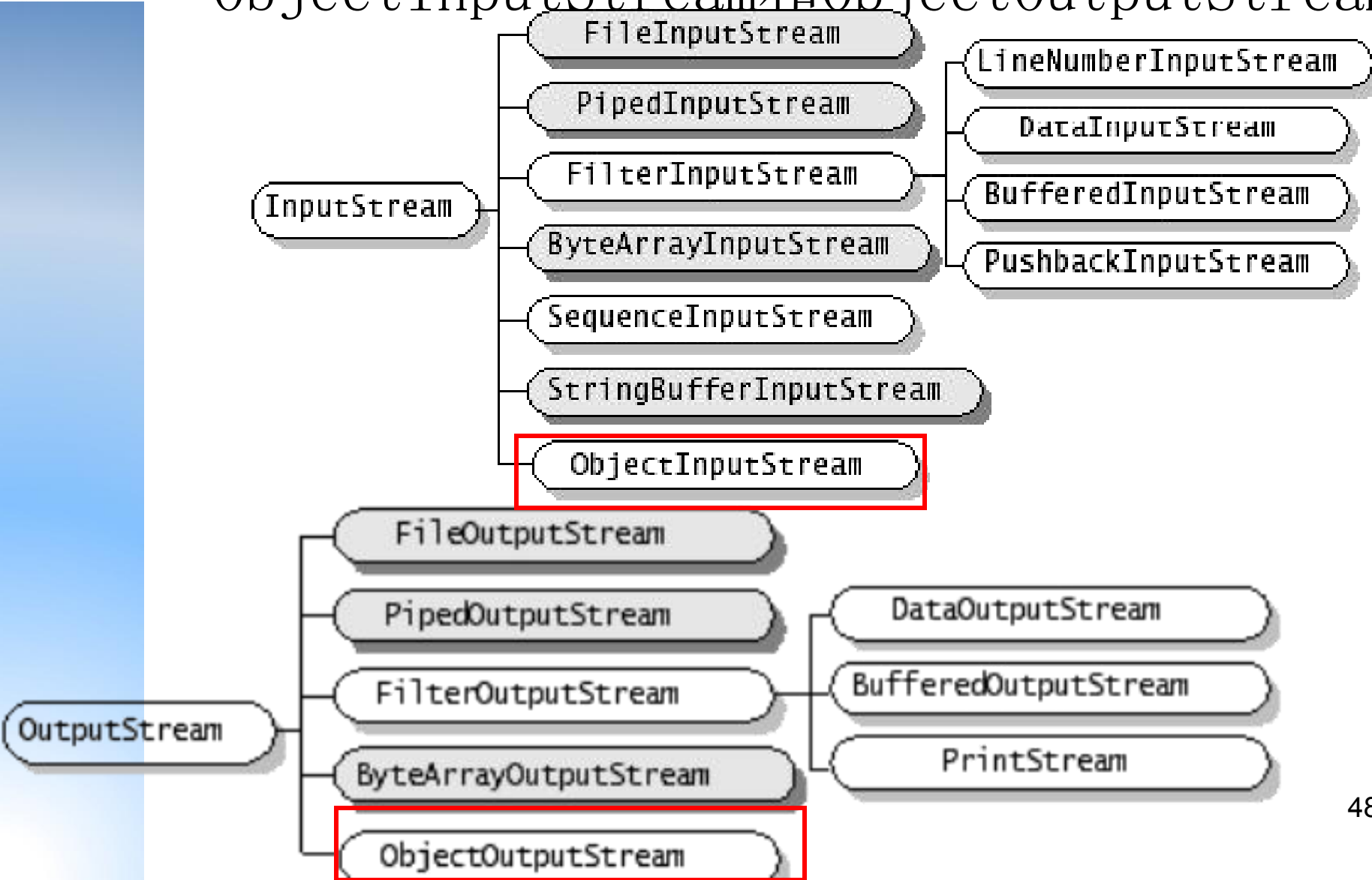
字节流

- BufferedInputStream和BufferedOutputStream
 - 缓冲区默认大小512字节
 - 当缓冲满或者flush方法调用时，缓冲流写入
 - 提高前面例子的效率
 - `DataOutputStream output = new DataOutputStream(new BufferedOutputStream(new FileOutputStream("temp.dat")))`
 - 应该使用缓冲区I/O来加速输入和输出。



字节流

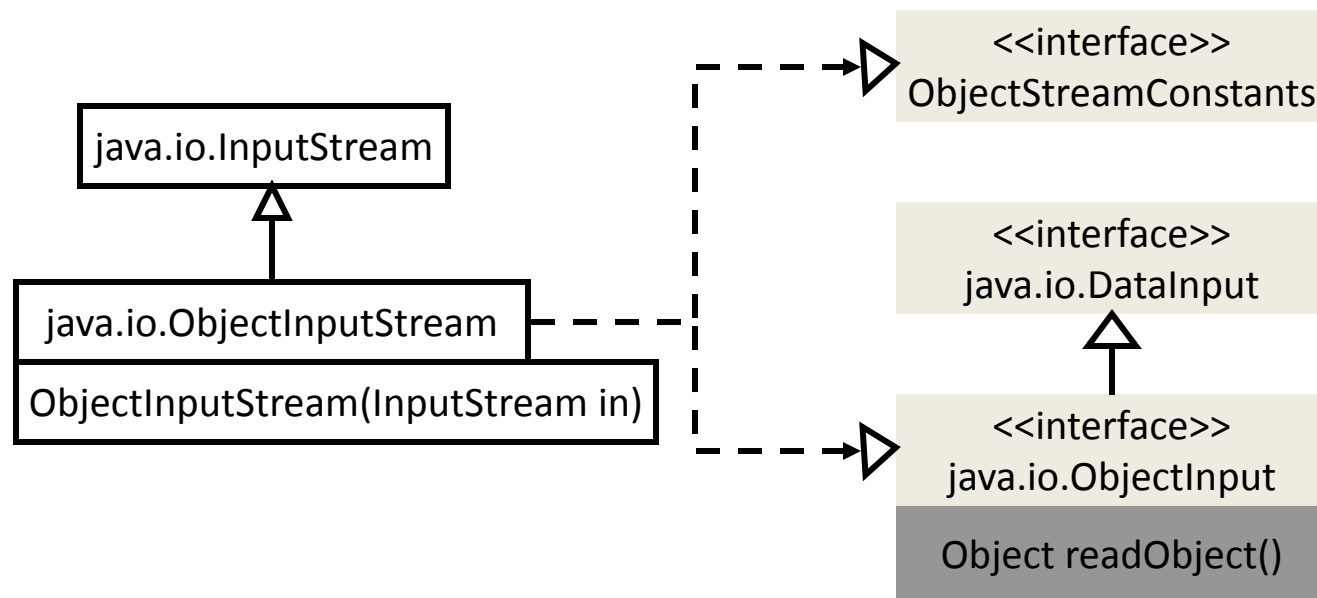
- ObjectInputStream和ObjectOutputStream





字节流

- ObjectInputStream和ObjectOutputStream





字节流

- `ObjectInputStream`和`ObjectOutputStream`
 - `DataInputStream`能处理的`ObjectInputStream`都能处理
 - 可以包装`InputStream`和`OutputStream`

```
public ObjectInputStream(InputStream in)
Public ObjectOutputStream(OutputStream out)
```



字节流

- ObjectOutputStream和ObjectOutputStream

```
import java.io.*;

public class TestObjectOutputStream {
    public static void main(String[] args) throws IOException{
        ObjectOutputStream output = new ObjectOutputStream(
            new BufferedOutputStream(new FileOutputStream("object.dat")));
        output.writeUTF("John");
        output.writeDouble(85.5);
        output.writeObject(new java.util.Date());
        output.close();
    }
}
```



字节流

- ObjectInputStream和ObjectOutputStream

```
import java.io.*;
public class TestObjectInputStream {
    public static void main(String[] args)
        throws ClassNotFoundException, IOException {
        ObjectInputStream input = new ObjectInputStream(
            new BufferedInputStream(new FileInputStream("object.dat")));
        String name = input.readUTF();
        double score = input.readDouble();
        java.util.Date date = (java.util.Date) (input.readObject());
        System.out.println(name + " " + score + " " + date);
        input.close();
    }
}
```

John 85.5 Mon Sep 03 10:34:54 CST 2012



字节流

- `ObjectInputStream`和`ObjectOutputStream`
 - 从`ObjectInputStream`读回对象时，必须与写入时的类型**顺序一致**
 - 要使用读回的对象，必须使用java安全类型转换



字节流

- 对象流的可序列化接口Serializable
 - 并不是每个对象都可以写到输出流，可以写到输出流中的对象称为可序列化的
 - 可序列化对象实现了 `java.io.Serializable` 接口
 - Serializable 接口是一种标记性接口，没有方法
 - 实现这个接口可以启动 java 的序列化机制，自动完成
 - Java API 中许多类都实现了 Serializable 接口。
。 `java.util.Date` 以及所有的 Swing GUI 组件



字节流

- 序列化数组

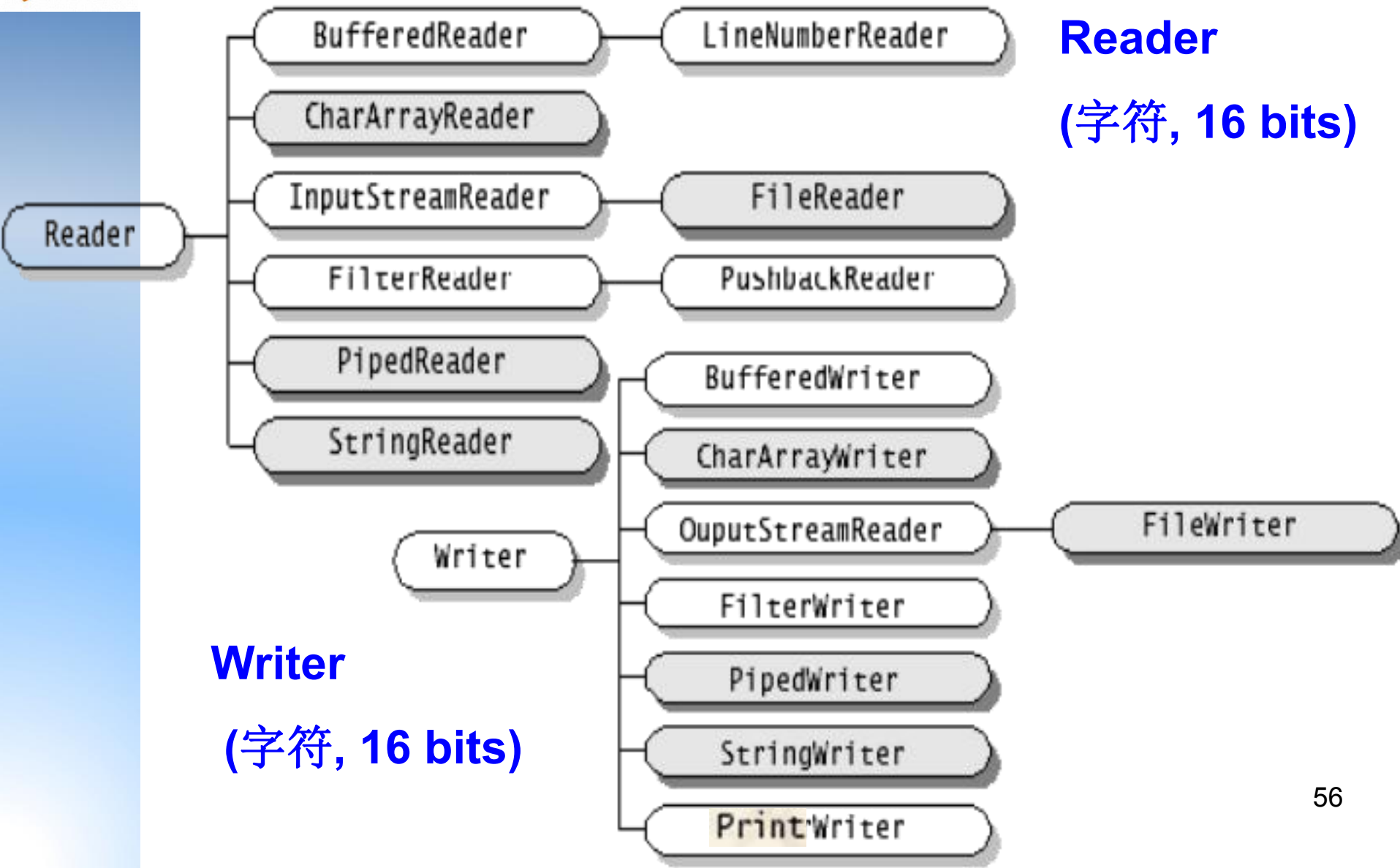
```
import java.io.*;
public class TestObjectStreamForArray {
    public static void main(String[] args) throws ClassNotFoundException, IOException{
        int[] numbers = {1,2,3,4,5};
        String[] strings = {"John","Jim","Jake"};
        ObjectOutputStream output = new ObjectOutputStream(
            new FileOutputStream("array.dat",true));
        output.writeObject(numbers);
        output.writeObject(strings);
        output.close();

        ObjectInputStream input = new ObjectInputStream(
            new FileInputStream("array.dat"));
        int[] newnumbers = (int[]) (input.readObject());
        String[] newstrings = (String[]) (input.readObject());
        for(int i=0;i<newnumbers.length;i++)
            System.out.print(newnumbers[i]+" ");
        System.out.println();
        for(int i=0;i<newstrings.length;i++)
            System.out.print(newstrings[i]+" ");
    }
}
```

1	2	3	4	5
John	Jim	Jake		



3. 字符流及其方法





字节流与字符流类型比较

来源与去处:Java 1.0类	相应的Java 1.1类
InputStream	Reader 适配器: InputStreamReader
OutputStream	Writer 适配器: OutputStreamWriter
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream (已弃用)	StringReader
(无相应的类)	StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter
PipedInputStream	PipedReader
PipedOutputStream	PipedWriter



字节流与字符流类型比较

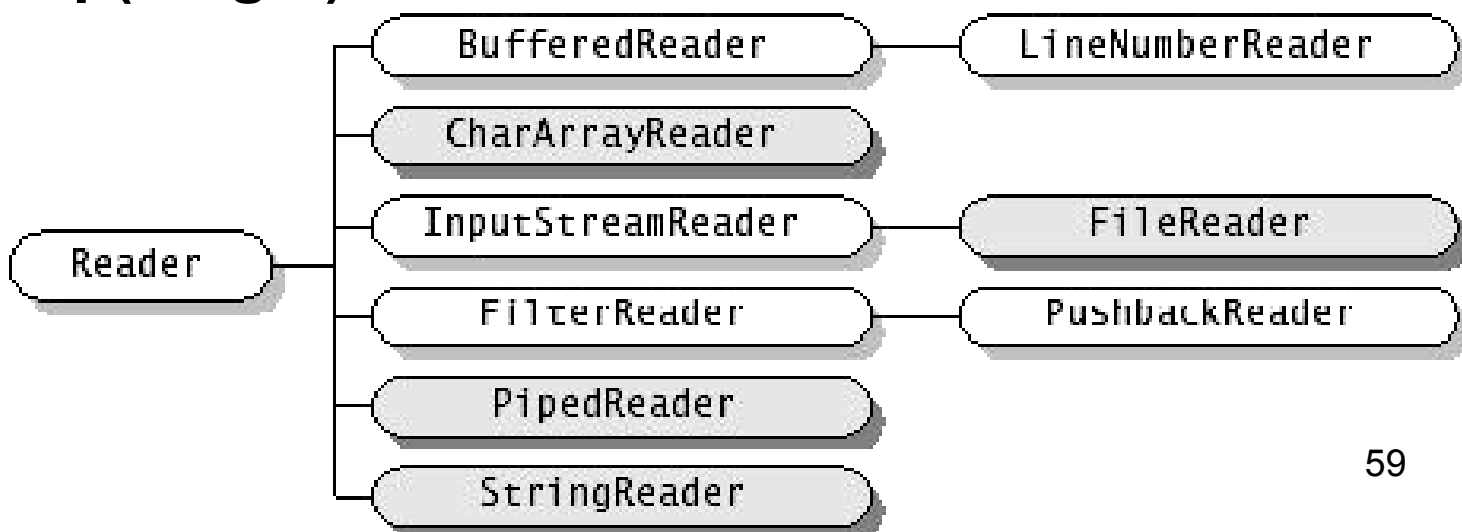
过滤器:Java 1.0类	相应的Java 1.1类
FilterInputStream	FilterReader
FilterOutputStream	FilterWriter (抽象类, 没有子类)
BufferedInputStream	BufferedReader (也有 readLine())
BufferedOutputStream	BufferedWriter
DataInputStream	使用 DataInputStream (当你需要使用 readLine() , 应该使用 BufferedReader)
PrintStream	PrintWriter
LineNumberInputStream	LineNumberReader
StreamTokenizer	StreamTokenizer (使用接受 Reader 的构造器)
PushBackInputStream	PushBackReader



Reader的方法

Reader的方法 (throws IOException)

- **int read()**
- **int read(char[] cbuf)**
- **int read(char[] cbuff, int offset, int length)**
- **void close()**
- **long skip(long n)**

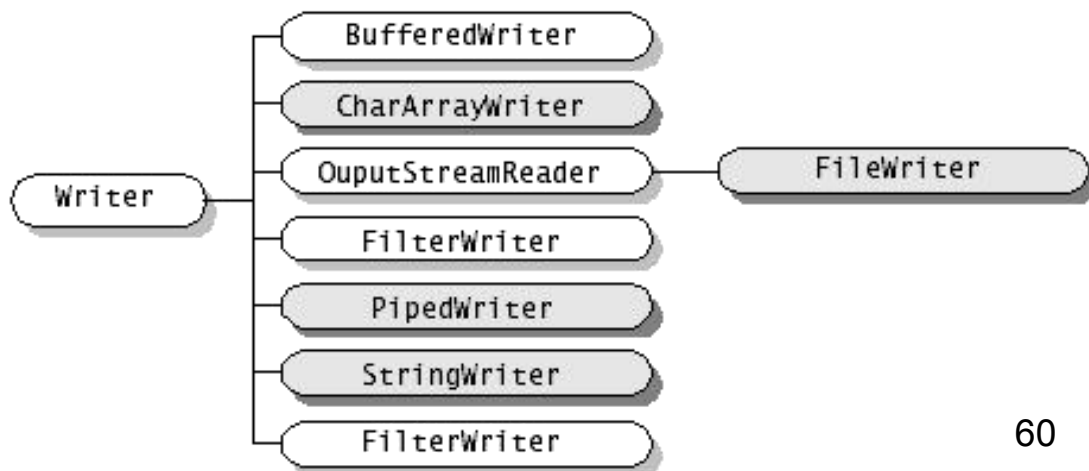




Writer的方法

Writer的方法 (throws IOException)

- **void write (int c)**
- **void write (char[] cbuf)**
- **void write (char[] cbuf, int offset, int length)**
- **void write (String string)**
- **void write (String string, int offset, int length)**
- **void close()**
- **void flush()**





8.3 标准I/O流

标准I/O (与文件File)

➤三个标准I/O流

- ***System.in*** : `InputStream`类的对象实例in作为标准输入流对象，对应于键盘输入
- ***System.out***: `PrintStream`类的对象实例out作为标准输出流对象，对应于显示器输出
- ***System.err***: `PrintStream`类的对象实例err作为标准错误输出流对象，对应于显示器输出



示例：标准I/O操作实现回音

```
import java.io.*;
public class Echo {
    public static void main(String[] args)
        throws IOException {
        BufferedReader stdin = new BufferedReader(
            new InputStreamReader(System.in));
        String s;
        while((s = stdin.readLine()) != null && s.length() != 0)
            System.out.println(s);
        // An empty line or Ctrl-Z terminates the program
    }
}
```



示例：标准I/O重定向实现文件复制

```
import java.io.*;
public class Redirecting {
    public static void main(String[] args)
        throws IOException {
        PrintStream consoleout = System.out;
        InputStream consolein = System.in
        BufferedInputStream in = new BufferedInputStream(
            new FileInputStream("Redirecting.java"));
        PrintStream out = new PrintStream(
            new BufferedOutputStream(
                new FileOutputStream("test.out")));
```



示例：标准I/O重定向实现文件复制

```
System.setIn(in);
System.setOut(out);
System.setErr(out);
BufferedReader br = new BufferedReader(
    new InputStreamReader(System.in));
String s;
while((s = br.readLine()) != null)
    System.out.println(s);
out.close(); // Remember this!
System.setOut(consoleout);
in.close();
System.setIn(consloein)
```

```
}
}
```




8.4 随机访问文件

RandomAccessFile

- 适用于由大小已知的记录组成的文件，可以使用 `seek()` 将记录从一处转移到另一处
- 不是继承自 `InputStream` 和 `OutputStream`，而是像 `DataInputStream` 和 `DataOutputStream` 一样实现了 `DataInput` 和 `DataOutput` 接口
- 类似于把 `DataInputStream` 和 `DataOutputStream` 结合在一起使用
- 只有 `RandomAccessFile` 类在文件上支持搜寻方法



8.4 随机访问文件

构造器和方法:

- **RandomAccessFile(File file, String mode)**
- **RandomAccessFile(String name, String mode)**

- **void seek(long pos)**
// 用于文件内移至新位置
- **long getFilePointer()**
// 得到当前的位置
- **long length()**
// 判断文件的大小



示例：随机访问文件

```
import java.io.*;

public class UsingRandomAccessFile {
    static String file = "rtest.dat";

    static void display() throws IOException {
        RandomAccessFile rf = new RandomAccessFile(file, "r");
        for(int i = 0; i < 7; i++)
            System.out.println("Value " + i + ": " + rf.readDouble());
        System.out.println(rf.readUTF());
        rf.close();
    }
}
```



```
public static void main(String[] args) throws IOException
{
    RandomAccessFile rf = new RandomAccessFile(file, "rw");
    for(int i = 0; i < 7; i++)
        rf.writeDouble(i*1.414);
    rf.writeUTF("The end of the file");
    rf.close();
    display();
    rf = new RandomAccessFile(file, "rw");
    rf.seek(5*8);
    System.out.println(rf.length());
    System.out.println(rf.getFilePointer());
    rf.writeDouble(47.0001);
    rf.close(); display(); }}
```

```
Value 0: 0.0
Value 1: 1.414
Value 2: 2.828
Value 3: 4.242
Value 4: 5.656
Value 5: 7.069999999999999
Value 6: 8.484
The end of the file
77
40
Value 0: 0.0
Value 1: 1.414
Value 2: 2.828
Value 3: 4.242
Value 4: 5.656
Value 5: 47.0001
Value 6: 8.484
The end of the file
```



压缩

- `ZipOutputStream` 压缩数据，生成Zip格式文件
- `GZIPOutputStream` 生成GZip格式文件
- `ZipInputStream` 解压缩已经生成的ZIP文件
- `GZIPInputStream` 解压缩已经生成的GZIP文件
- 继承于`InputStream`和`OutputStream`



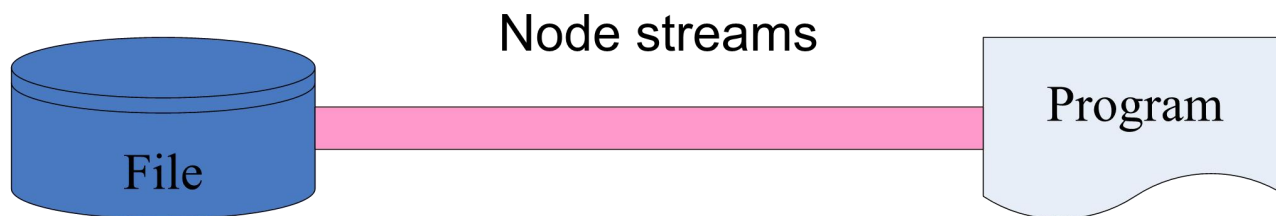
4. IO流

- 节点流
- 处理流
- 缓冲流
- 打印流
- 数据流
- 对象流

(1) 节点流

➤节点流

- 包含从特殊位置读、写的基本功能
- 节点流的包括文件，内容和管道





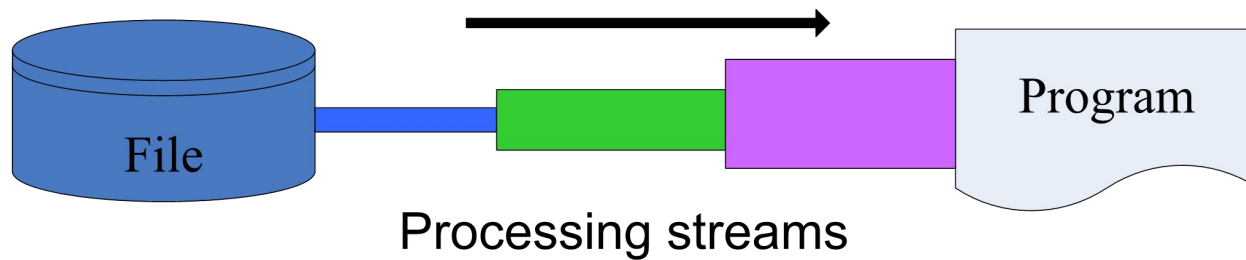
(1) 节点流

类型	字节流	字符流
文件	FileInputStream FileOutputStream	FileReader FileWriter
内存数组	ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
内存字符串	----- -----	StringReader StringWriter
管道	PipedInputStream PipedOutputStream	PipedReader PipedWriter

(2) 处理流

➤ 处理流

- 结合在其他流之上
- 修改或管理流中数据， 并提供额外的功能





(2) 处理流

字符	字节流	字符流
缓冲	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
过滤	FilterInputStream FilterOutputStream	FilterReader FilterWriter
字节与字符间 转换	----- -----	InputStreamReader OutputStreamWriter
数据转换	DataInputStream DataOutputStream	----- -----
对象序列化	ObjectInputStream ObjectOutputStream	----- -----
打印	PrintStream	PrintWriter



(3) 缓冲流

- 没有缓冲的I/O，直接读写效率低，对硬盘损害大
- 带缓冲的输入流从一个类似于缓冲区的内存区域中读取数据，当缓冲区为空时，调用基本的输入API；同样地，缓冲输出流向缓冲区中写数据，在缓冲区已满时调用基本的输出API
- 嵌套在其他节点流之上，对读写数据提供缓冲功能，提高了读写的效率，增加了一些新的方法



(3) 缓冲流

- **BufferedReader(Reader in)**
- **BufferedReader (Reader in, int sz)**
- **BufferedWriter(Writer out)**
- **BufferedWriter(Writer out, int sz)**
- **BufferedInputStream(InputStream in)**
- **BufferedInputStream(InputStream in, int size)**
- **BufferedOutputStream (OutputStream out)**
- **BufferedOutputStream(OutputStream out, int size)**



(3) 缓冲流

➤ **void mark()**

// 标记流中的当前位置.

➤ **void reset()**

// 尝试将该流重新定位到最近标记的点.

➤ **void readLine()**

// 仅适用于BufferedReader, 读文本中的一行

➤ **void newLine()**

// 仅适用于BufferedWriter, 写入一个行分隔符

➤ **void flush()**

// 刷新流



(4) 打印流

- **PrintStream** (字节)
- **PrintWriter** (字符)
 - 为其他输出流添加了功能，使它们能够方便地打印各种数据值表示形式。分别针对字节和字符，提供了重载的print和println方法用于多种数据类型的输出
- 不抛出**IOException**
- 自动的**Flush**



(4) 打印流

- **PrintWriter(Writer out)**
- **PrintWriter(File file)**
- **PrintWriter(OutputStream out)**
- **PrintWriter(String fileName)**
- **PrintStream(OutputStream out)**
- **PrintStream(File file)**
- **PrintStream(String fileName)**



(5) 数据流

数据流 (DataInputStream/DataOutputStream)

- 数据流提供了可以存取与机器无关的**java**原始类型的数据的方法，支持原始数据类型的输入输出，包括：**boolean, char, byte, short, int, long, float** 和 **double**类型
- 所有的数据流分别继承自**InputStream**和**OutputStream**,实现了**DataInput**和**DateOutput**接口
- 需要在**InputStream**和**OutputStream**类型的流上，也就是字节流之上进行构造
- 构造器
 - DataInputStream (InputStream in)
 - DataOutputStream (OutputStream out)



示例：数据流操作

```
import java.io.*;
public class TestDataStream {
    public static void main(String[] args) {
        ByteArrayOutputStream baos =
            new ByteArrayOutputStream();
        DataOutputStream dos =
            new DataOutputStream(baos);
        try {
            dos.writeDouble(Math.random());
            dos.writeBoolean(true);
```



示例：数据流操作

```
ByteArrayInputStream bais =  
    new ByteArrayInputStream(baos.toByteArray());  
System.out.println(bais.available());  
DataInputStream dis = new DataInputStream(bais);  
System.out.println(dis.readDouble());  
System.out.println(dis.readBoolean());  
dos.close();  
dis.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}  
}  
}
```

```
9  
0.9220506705720964  
true
```



(6) 对象流

- 对象流支持**Object**的输入输出
 - 像数据流一样支持对象的输入输出
 - 对象需要是可序列化的类型
- **ObjectInputStream**和**ObjectOutputStream**
 - 这些类实现了DataInput 和 DataOutput的子接口 **ObjectInput** 和 **ObjectOutput** 接口
 - 一个对象流可以包含基本类型和引用类型数据的混合



示例：对象流操作

```
import java.io.*;
import java.util.*;
public class Logon implements Serializable {
    private Date date = new Date();
    private String username;
    private String password;
    public Logon(String name, String pwd) {
        username = name;
        password = pwd;
    }
    public String toString() {
        return "logon info: \n  username: " + username +
            "\n  date: " + date + "\n  password: " + password;
    }
}
```



Java

示例：对象流操作

```
public static void main(String[] args) throws Exception {  
    Logon a = new Logon("Hulk", "myLittlePony");  
    System.out.println("logon a = " + a);  
    ObjectOutputStream o = new ObjectOutputStream(  
        new FileOutputStream("Logon.out"));  
    o.writeObject(a);  
    o.close();  
    ObjectInputStream in = new ObjectInputStream(  
        new FileInputStream("Logon.out"));  
    System.out.println("Recovering object at " + new Date());  
    a = (Logon)in.readObject();  
    System.out.println("logon a = " + a);  
}  
}
```

```
logon a = logon info:  
    username: Hulk  
    date: Fri Aug 31 18:11:01 CST 2012  
    password: myLittlePony  
Recovering object at Fri Aug 31 18:11:01 CST 2012  
logon a = logon info:  
    username: Hulk  
    date: Fri Aug 31 18:11:01 CST 2012  
    password: myLittlePony
```



Scanner

- Java5 添加了java.util.Scanner
- 使用正则表达式,正则表达式通常被用来检索、替换那些符合某个模式(规则)的文本。
- `public boolean hasNext(Pattern pattern)`
- `public boolean hasNextInt()`
- `public String next();`
- `public int nextInt();`



Scanner

```
import java.util.Scanner;
public class ScannerDemo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // 从键盘接收数据
        // next方式接收字符串
        System.out.println("next方式接收: ");
        // 判断是否还有输入
        if (scan.hasNext()) {
            String str1 = scan.next();
            System.out.println("输入的数据为: " + str1);
        }
        scan.close();
    }
}
```

next方式接收:
java scanner
输入的数据为: java



Scanner

```
import java.util.Scanner;
public class ScannerDemo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // 从键盘接收数据
        // nextLine方式接收字符串
        System.out.println("nextLine方式接收: ");
        // 判断是否还有输入
        if (scan.hasNextLine()) {
            String str2 = scan.nextLine();
            System.out.println("输入的数据为: " + str2);
        }
        scan.close();
    }
}
```

nextLine方式接收:

java scanner

输入的数据为: java scanner



Scanner

```
import java.util.Scanner;
public class ScannerDemo {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        // 从键盘接收数据
        int i = 0;
        System.out.print("输入整数: ");
        if (scan.hasNextInt()) { // 判断输入的是否是整数
            i = scan.nextInt(); // 接收整数
            System.out.println("整数数据: " + i);
        }
    }
}
```

输入整数: 12

整数数据: 12



Scanner

```
public static void main(String[] args) {  
    Scanner s = new Scanner("123 asdf sd 45 789 sdf asdf, sdf.sdf, asdf .....asdfkl las");  
    //s.useDelimiter(" |,|\\.");  
    while (s.hasNext()) {  
        System.out.println(s.next());  
    }  
}
```

```
123  
asdf  
sd  
45  
789  
sdf  
asdf, sdf.sdf, asdf  
.....asdfkl  
las
```

```
123  
asdf  
sd  
45  
789  
sdf  
asdf  
sdf  
sdf  
asdf
```

```
asdfkl
```

```
las
```



本章小结

◆ 流与相关类

- ◆ InputStream/OutputStream

- ◆ Reader/Writer

◆ 标准I / O流

- ◆ System.in

- ◆ System.out

- ◆ System.err

◆ 文件输入输出流

- ◆ FileInputStream/FileOutputStream

- ◆ FileReader/Writer

◆ 随机访问文件

- ◆ RandomAccessFile