

输入/输出处理

描述java.io包的主要特性

流， 输入流， 输出流

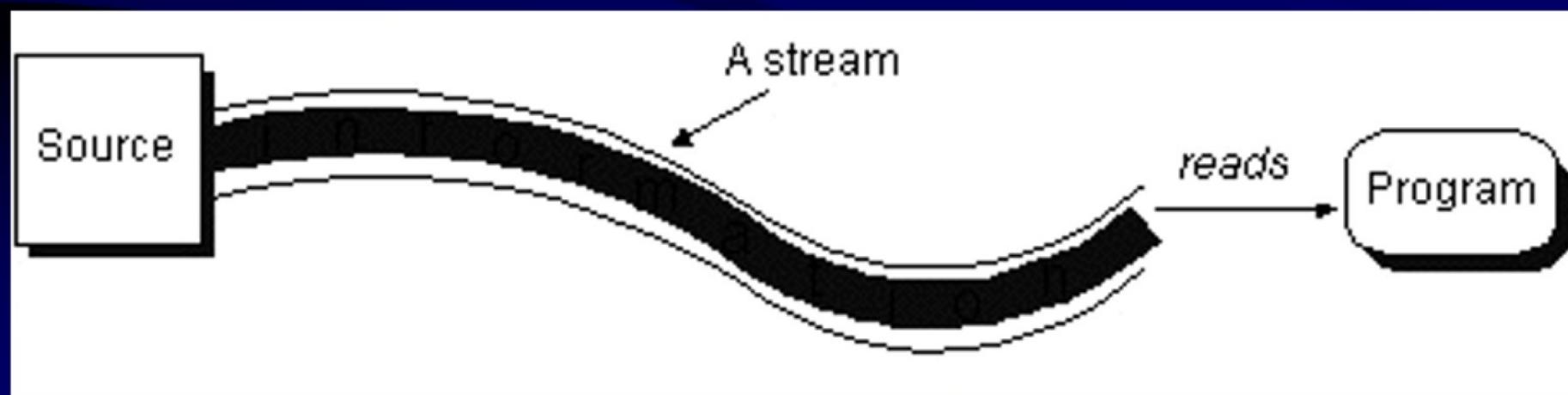
文件流的相关操作

对象的串化

输入/输出处理

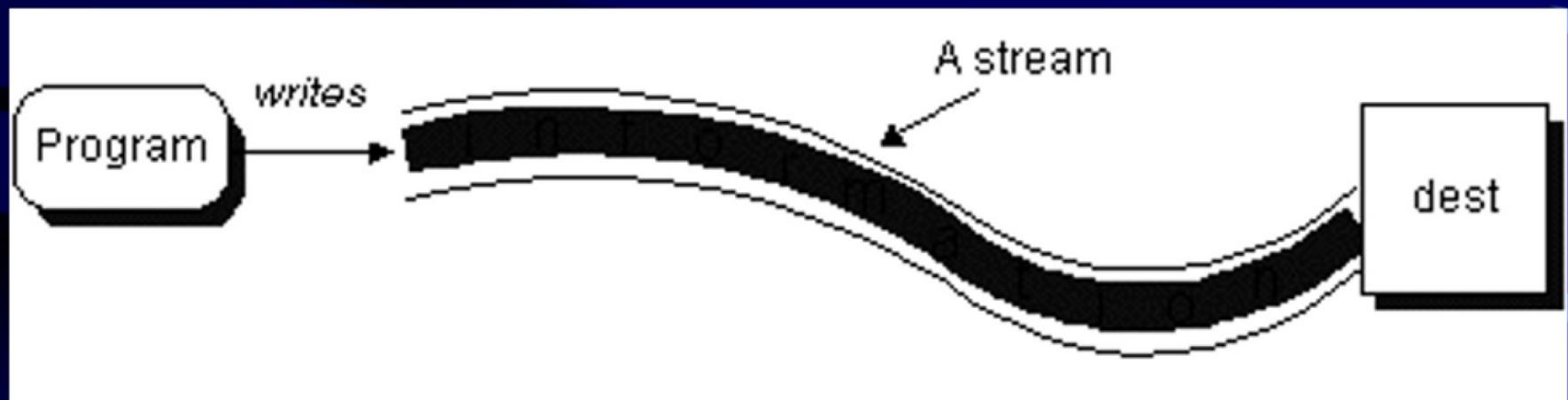
流

程序对于信息的读取是个流动的过程。
file, memory, a socket。

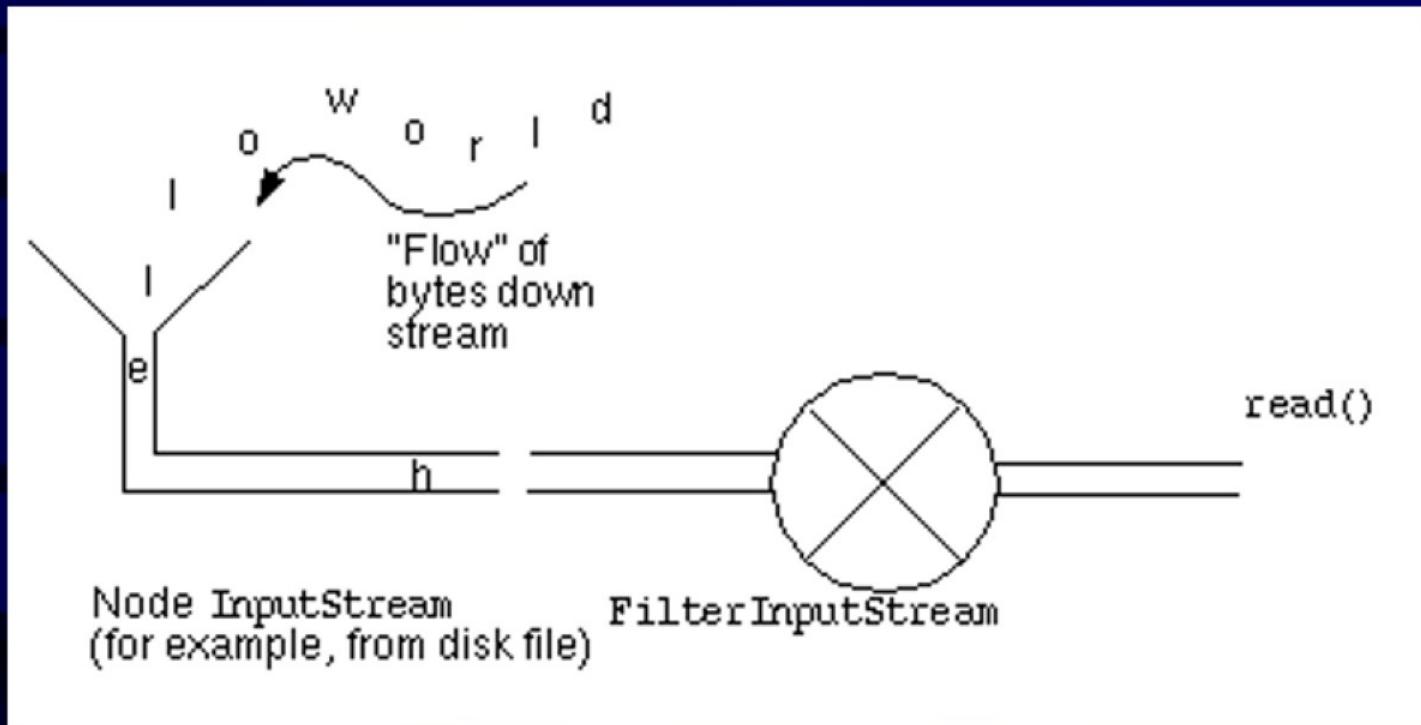


输入/输出处理

程序对于信息的处理也是个流动的过程



输入/输出处理



输入/输出处理--- java.io包

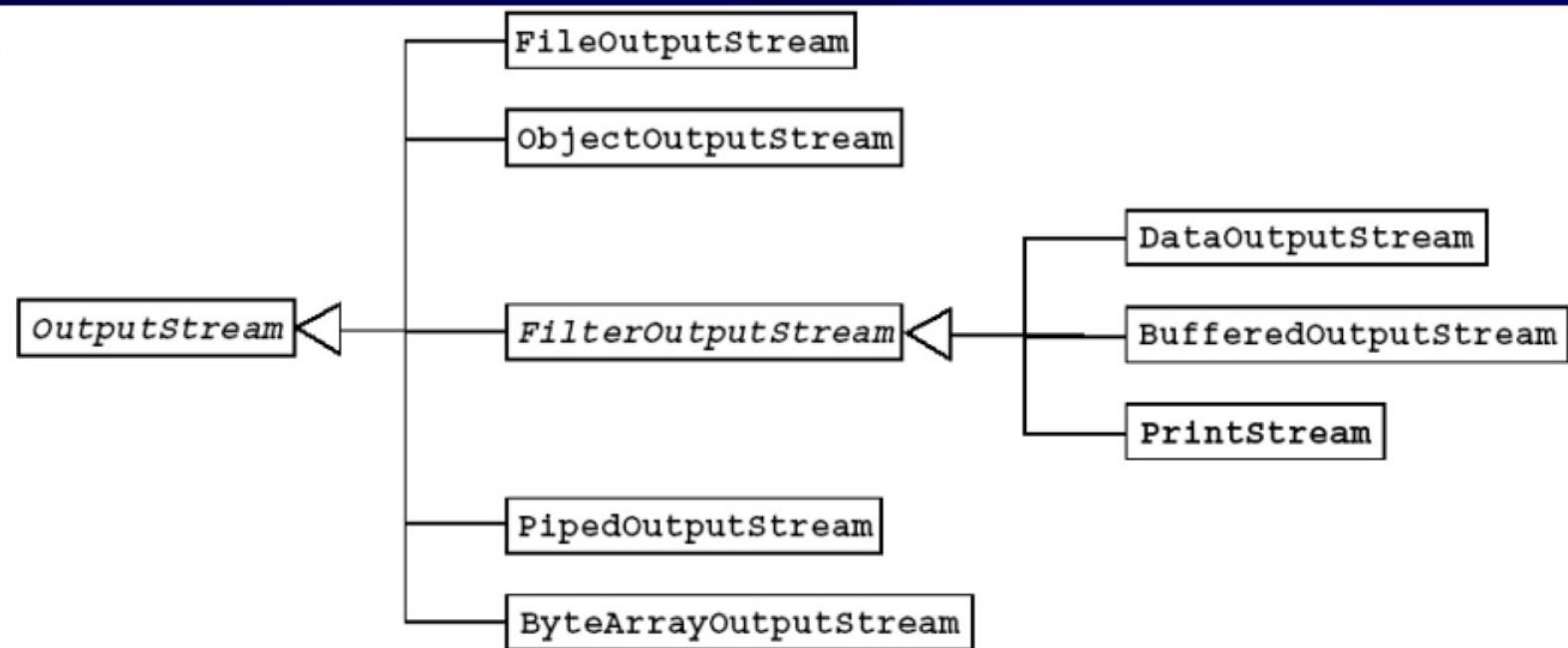
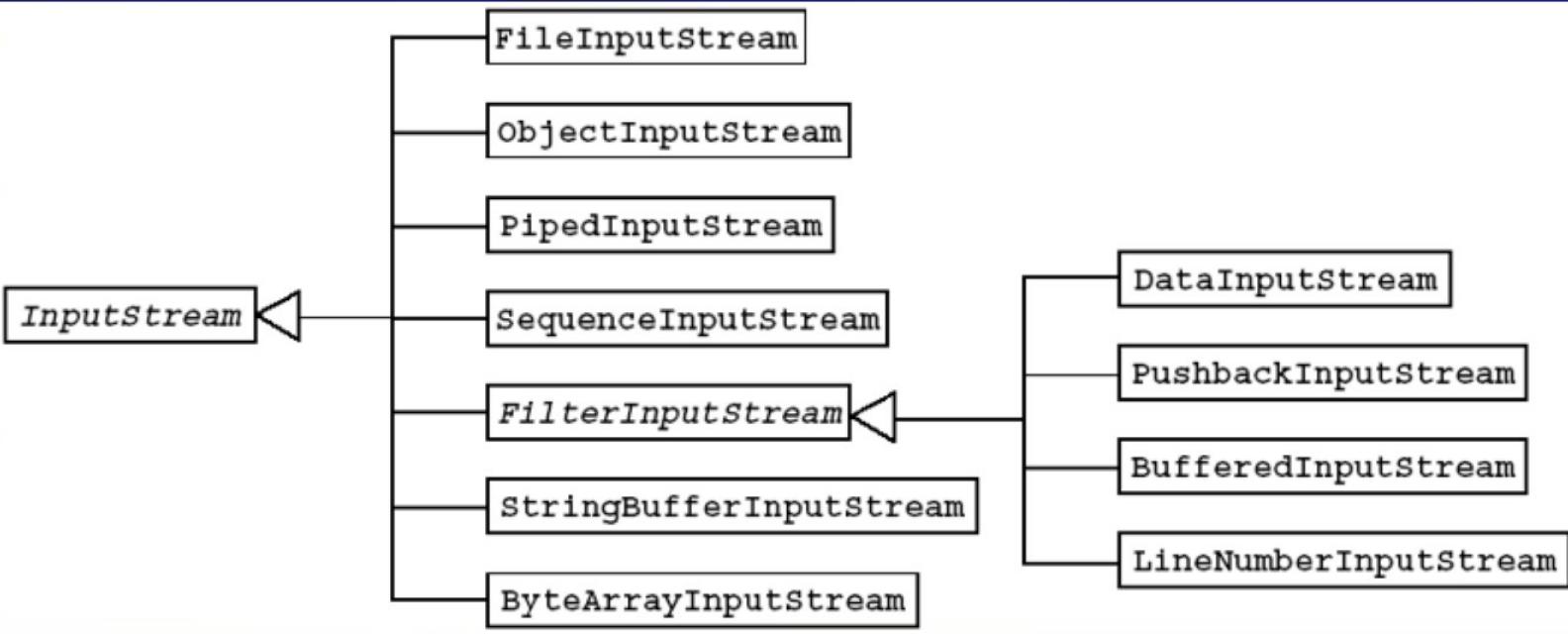
- 字节流 → stream

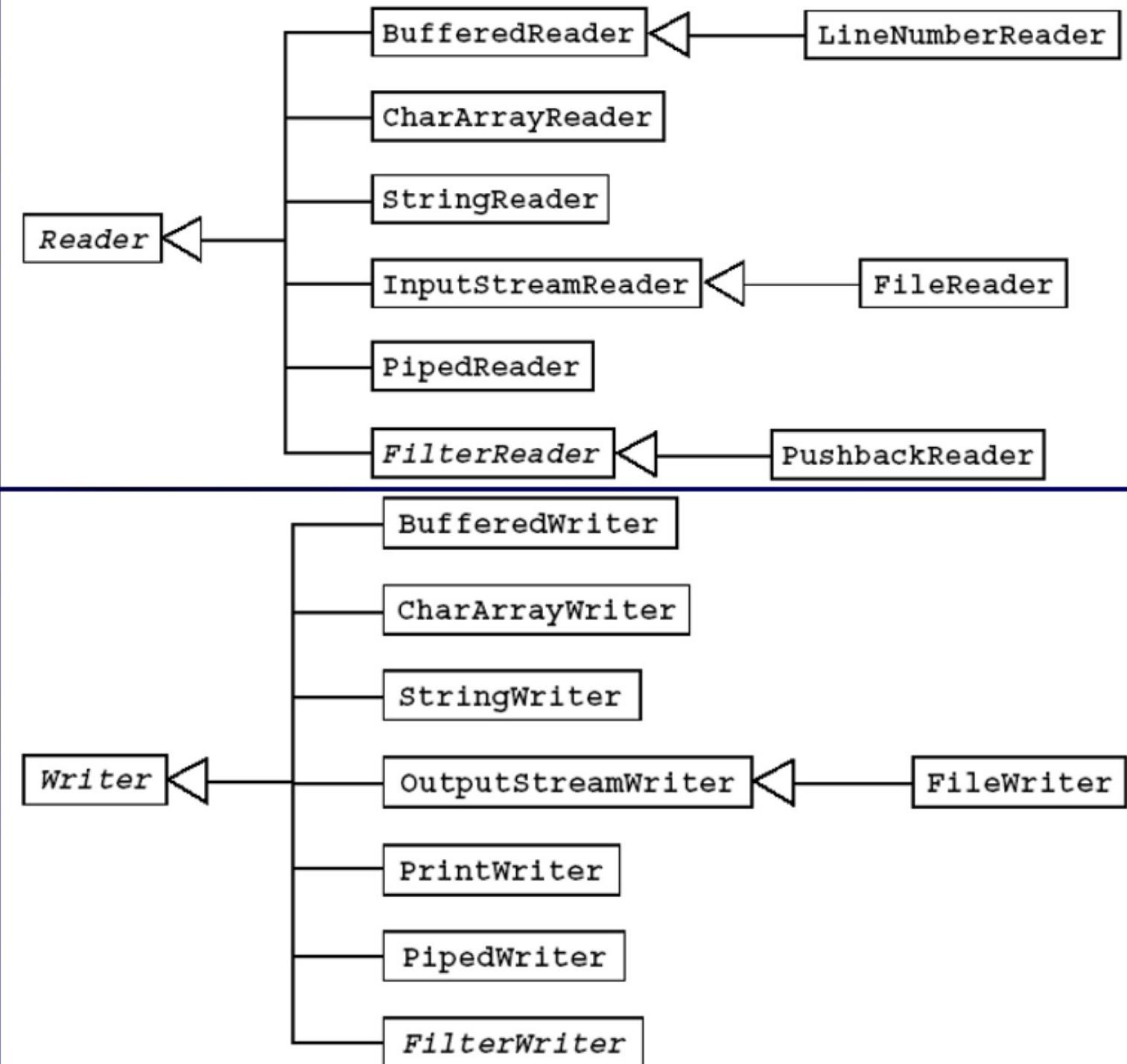
从InputStream和OutputStream派生出来的一系列类。
这类流以字节(byte)为基本处理单位。

- 字符流 → reader & writer

从Reader和Writer派生出的一系列类，这类流以16位的Unicode码表示的字符为基本处理单位。

	byte streams	character streams
source streams	InputStream	Reader
sink streams	OutputStream	Writer





输入/输出处理--- java.io包

- 文件处理

File、 RandomAccessFile;

- 接口 → interface

DataInput、 DataOutput、 ObjectInput、
ObjectOutput;

输入/输出处理--- InputStream

- 从流中读取数据

```
int read( );
```

```
int read( byte b[ ] );
```

```
int read( byte b[ ], int off, int len );
```

这三个方法提供对输入管道数据的存取。简单读方法返回一个int值，它包含从流里读出的一个字节或者-1，其中后者表明文件结束。其它两种方法将数据读入到字节数组中，并返回所读的字节数。第三个方法中的两个int参数指定了所要填入的数组的子范围。

输入/输出处理--- InputStream

- `int available();`

这个方法报告立刻可以从流中读取的字节数。在这个调用之后的实际读操作可能返回更多的字节数。

- `long skip(long n);`

这个方法丢弃了流中指定数目的字符。

- `close();`

完成流操作之后，就关闭这个流。如果有一个流所组成的栈，使用过滤器流，就关闭栈顶部的流。这个关闭操作会关闭其余的流。

输入/输出处理--- InputStream

- 使用输入流中的标记

```
void mark( int readlimit );
```

```
void reset( );
```

```
boolean markSupported( );
```

如果流支持“回放”操作，则这些方法可以用来完成这个操作。如果mark()和reset()方法可以在特定的流上操作，则markSupported()方法将返回ture。

mark(int)方法用来指明应当标记流的当前点和分配一个足够大的缓冲区，它最少可以容纳参数所指定数量的字符。在随后的read()操作完成之后，调用reset()方法来返回你标记的输入点。

输入/输出处理--- OutputStream

- 输出数据

- `void write(int b);`

- `void write(byte b[]);`

- `void write(byte b[], int off, int len);`

- `flush()`

- 刷空输出流，并输出所有被缓存的字节。

- 有时一个输出流在积累了若干次之后才进行真正的写操作。`flush()`方法允许你强制执行写操作。

- 关闭流

- `close();`

输入/输出处理---文件处理

- File、FileInputStream、 FileOutputStream、 RamdomAccessFile
- 类File提供了一种与机器无关的方式来描述一个文件对象的属性。
- 文件的生成

public File(String path);

public File(String path, String name);

public File(File dir, String name);

输入/输出处理---文件处理

```
File myFile;
```

```
myFile = new File("myfile.txt");
```

```
myFile = new File("MyDocs", "myfile.txt");
```

```
File myDir = new File("MyDocs");
```

```
myFile = new File(myDir, "myfile.txt");
```

输入/输出处理---文件处理

- 文件描述
- 文件名的处理

String getName();//得到一个文件的名称（不包括路径）

String getPath();//得到一个文件的路径名

String getAbsolutePath();//得到一个文件的绝对路径名

String getParent();//得到一个文件的上一级目录名

String renameTo(File newName);//将当前文件名更名为给定文件的完整路径

输入/输出处理---文件处理

- 文件属性测试

boolean exists();//测试当前File对象所指示的文件是否存在

boolean canWrite();//测试当前文件是否可写

boolean canRead();//测试当前文件是否可读

boolean isFile();//测试当前文件是否是文件（不是目录）

boolean isDirectory();//测试当前文件是否是目录

输入/输出处理---文件处理

- 普通文件信息和工具

long lastModified();//得到文件最近一次修改的时间

long length();//得到文件的长度，以字节为单位

boolean delete();//删除当前文件

- 目录操作

boolean mkdir();//根据当前对象生成一个由该对象指定的路径

String list();//列出当前目录下的文件

输入/输出处理---文件处理

- 文件的顺序处理
- 类FileInputStream和FileOutputStream用来进行文件I/O处理，由它们所提供的方法可以打开本地主机上的文件，并进行顺序的读/写。

输入/输出处理---文件处理

```
FileInputStream fis;  
try{  
    fis = new FileInputStream( "text" );  
    System.out.print( "content of text is : " );  
    int b;  
    while( (b=fis.read())!=-1 ){  
        System.out.print( (char)b );  
    }  
}catch( FileNotFoundException e ){  
    System.out.println( e );  
}catch( IOException e ){  
    System.out.println( e );  
}
```

输入/输出处理---文件处理

- 随机存取文件 →
- public class RandomAccessFile
extends Object
implements DataInput, DataOutput
 - 接口DataInput中定义的方法主要包括从流中读取基本类型的数据、读取一行数据、或者读取指定长度的字节数。如：readBoolean()、readInt()、readLine()、readFully()等。
 - 接口DataOutput中定义的方法主要是向流中写入基本类型的数据、或者写入一定长度的字节数组。如：writeChar()、writeDouble()、write()等。

输入/输出处理---文件处理

用如下两种方法来打开一个随机存取文件：

用文件名

```
myRAFile = new RandomAccessFile(String name, String mode);
```

用文件对象

```
myRAFile = new RandomAccessFile(File file, String mode);
```

mode参数决定了你对这个文件的存取是只读(r)还是读/写(rw)。

可以打开一个数据库文件并准备更新：

```
RandomAccessFile myRAFile;
```

```
myRAFile = new RandomAccessFile("db/stock.dbf","rw");
```

输入/输出处理---文件处理

文件指针的操作

- long getFilePointer();
- void seek(long pos);
- int skipBytes(int n);

可以使用随机存取文件来得到文件输出的添加模式。

```
myRAFile = new RandomAccessFile("java.log","rw");
```

```
myRAFile.seek(myRAFile.length());
```

```
// Any subsequent write()s will be appended to the file
```

输入/输出处理---过滤流

- 过滤流在读/写数据的同时可以对数据进行处理，它提供了同步机制，使得某一时刻只有一个线程可以访问一个I/O流，以防止多个线程同时对一个I/O流进行操作所带来的意想不到的结果。
- 类FilterInputStream和FilterOutputStream分别作为所有过滤输入流和输出流的父类。

输入/输出处理---过滤流

- 为了使用一个过滤流，必须首先把过滤流连接到某个输入/出流上，通常通过在构造方法的参数中指定所要连接的输入/出流来实现。例如：

```
FilterInputStream( InputStream in );
```

```
FilterOutputStream( OutputStream out );
```

输入/输出处理---过滤流

- `BufferedInputStream`和`BufferedOutputStream`缓冲流，用于提高输入/输出处理的效率。

`java.lang.Object`

|

+----`java.io.InputStream`

|

+----`java.io.FilterInputStream`

|

+----`java.io.BufferedInputStream`

输入/输出处理---过滤流

- DataInputStream 和 DataOutputStream

可以以与机器无关的格式读取各种类型的数据。

```
public class DataInputStream  
extends FilterInputStream  
implements DataInput
```

输入/输出处理---过滤流

DataInputStream

- byte readByte()
- long readLong()
- double readDouble()

DataOutputStream

- void writeByte(byte)
- void writeLong(long)
- void writeDouble(double)

输入/输出处理---过滤流

- **LineNumberInputStream**

除了提供对输入处理的支持外，
LineNumberInputStream可以记录当前的行号。

- **PushbackInputStream**

提供了一个方法可以把刚读过的字节退回到输入流中。

- **PrintStream**

打印流的作用是把Java语言的内构类型以其字符表示形式送到相应的输出流。

输入/输出处理--- I/O异常

- 进行I/O操作时可能会产生I/O异常，属于非运行时异常，应该在程序中处理。
如：FileNotFoundException, EOFException,
IOException

输入/输出处理---流结束的判断

- `read()`
 返回-1。
- `readXXX()`
 `readInt()`、`readByte()`、`readLong()`、`readDouble()`等；
 产生`EOFException`。
- `readLine()`
 返回`null`。

输入/输出处理---字符流

- java.io包中提供了专门用于字符流处理的类（以Reader和Writer为基础派生出的一系列类）。
- Reader和Writer

这两个类是抽象类，只是提供了一系列用于字符流处理的接口，不能生成这两个类的实例，只能通过使用由它们派生出来的子类对象来处理字符流。

```
1 import java.io.*;
2
3 public class TestNodeStreams {
4     public static void main(String[] args) {
5         try {
6             FileReader input = new FileReader(args[0]);
7             FileWriter output = new FileWriter(args[1]);
8             char[] buffer = new char[128];
9             int charsRead;
10
11            // read the first buffer
12            charsRead = input.read(buffer);
13
14            while ( charsRead != -1 ) {
15                // write the buffer out to the output file
16                output.write(buffer, 0, charsRead);
17
18                // read the next buffer
19                charsRead = input.read(buffer);
20            }
21
22            input.close();
23            output.close();
24        } catch (IOException e) {
25            e.printStackTrace();
26        }
27    }
28 }
```

输入/输出处理---字符流

- Reader类是处理所有字符流输入类的父类。

- 读取字符

```
public int read() throws IOException;
```

```
public int read(char cbuf[]) throws IOException;
```

```
public abstract int read(char cbuf[],int off,int len)  
throws IOException;
```

输入/输出处理---字符流

- 标记流

```
public boolean markSupported();
public void mark(int readAheadLimit) throws
    IOException;
public void reset() throws IOException;
- 关闭流
public abstract void close() throws IOException;
```

输入/输出处理---字符流

- Writer类是处理所有字符流输出类的父类。

- 向输出流写入字符

- `public void write(int c) throws IOException;`

- `public void write(char cbuf[]) throws IOException;`

- `public abstract void write(char cbuf[],int off,int len)`
`throws IOException;`

- `public void write(String str) throws IOException;`

- `public void write(String str,int off,int len)` throws
`IOException;`

输入/输出处理---字符流

- flush()

刷空输出流，并输出所有被缓存的字节。

- 关闭流

```
public abstract void close() throws IOException;
```

输入/输出处理---字符流处理

- InputStreamReader和OutputStreamWriter →
- java.io包中用于处理字符流的最基本的类,用来在字节流和字符流之间作为中介。

输入/输出处理---字符流处理

- 生成流对象

```
public InputStreamReader(InputStream in);
```

```
public InputStreamReader(InputStream in, String enc)  
throws UnsupportedEncodingException;
```

```
public OutputStreamWriter(OutputStream out);
```

```
public OutputStreamWriter(OutputStream out, String  
enc) throws UnsupportedEncodingException;
```

输入/输出处理---字符流

- 读入和写出字符

基本同Reader和Writer。

- 获取当前编码方式

```
public String getEncoding();
```

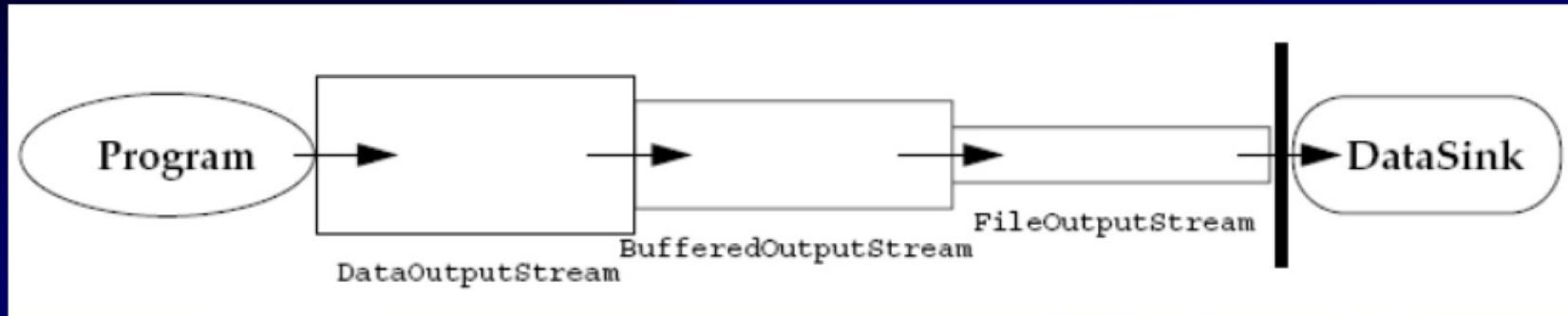
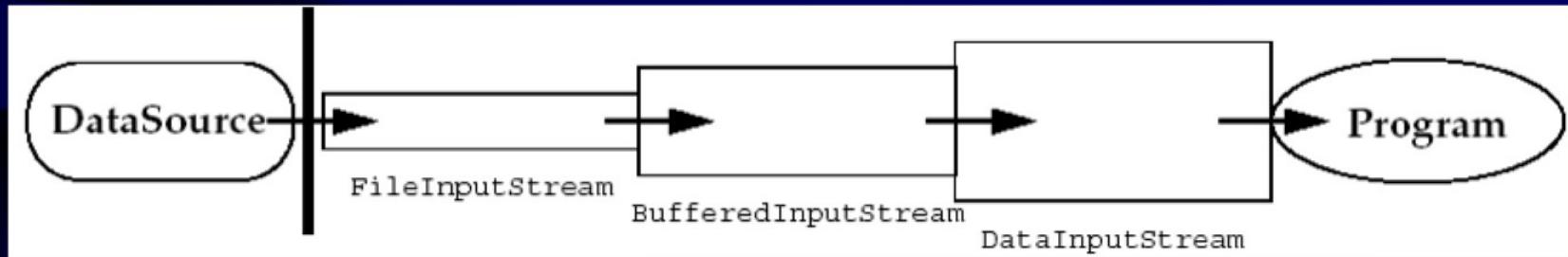
- 关闭流

```
public void close() throws IOException;
```

输入/输出处理---缓冲流

- BufferedReader和BufferedWriter →
- 生成流对象

```
public BufferedReader(Reader in);  
public BufferedReader(Reader in, int sz);  
public BufferedWriter(Writer out);  
public BufferedWriter(Writer out, int sz);
```



```
1 import java.io.*;
2
3 public class TestBufferedStreams {
4     public static void main(String[] args) {
5         try {
6             FileReader      input      = new FileReader(args[0]);
7             BufferedReader bufInput   = new BufferedReader(input);
8             FileWriter      output     = new FileWriter(args[1]);
9             BufferedWriter bufOutput = new BufferedWriter(output);
10            String line;
11
12            // read the first line
13            line = bufInput.readLine();
14
15            while ( line != null ) {
16                // write the line out to the output file
17                bufOutput.write(line, 0, line.length());
18                bufOutput.newLine();
19
20                // read the next line
21                line = bufInput.readLine();
22            }
23
24            bufInput.close();
25            bufOutput.close();
26        } catch (IOException e) {
27            e.printStackTrace();
28        }
29    }
30 }
```

输入/输出处理---缓冲流

- 读入/写出字符

除了Reader和Writer中提供的基本的读写方法外，增加对整行字符的处理。

```
public String readLine() throws IOException;  
public void newLine() throws IOException;
```

```
import java.io.*;
public class CharInput{
    public static void main(String args[]) throws
    FileNotFoundException,IOException{
        String s;
        FileInputStream is;
        InputStreamReader ir;
        BufferedReader in;
        is=new FileInputStream("test.txt");
        ir=new InputStreamReader(is);
        in=new BufferedReader(ir);
        while((s=in.readLine())!=null)
            System.out.println("Read: "+s);
    }
}
```

- 运行结果如下：

Read: java is a platform independent

Read: programming language

Read: it is a

Read: object oriented language.

输入/输出处理--- java.io包

Type	Character Streams	Byte Streams
File	FileReader FileWriter	FileInputStream FileOutputStream
Memory: Array	CharArrayReader CharArrayWriter	ByteArrayInputStream ByteArrayOutputStream
Memory: String	StringReader StringWriter	
Pipe	PipedReader PipedWriter	PipedInputStream PipedOutputStream

输入/输出处理---过滤流

Type	Character Streams	Byte Streams
Buffering	BufferedReader BufferedWriter	BufferedInputStream BufferedOutputStream
Filtering	<i>FilterReader</i> <i>FilterWriter</i>	<i>FilterInputStream</i> <i>FilterOutputStream</i>
Converting between bytes and character	InputStreamReader OutputStreamWriter	
Object serialization		ObjectInputStream ObjectOutputStream
Data conversion		DataInputStream DataOutputStream
Counting	LineNumberReader	LineNumberInputStream
Peeking ahead	PushbackReader	PushbackInputStream
Printing	PrintWriter	PrintStream

输入/输出处理---编码方式

- 注意：在读取字符流时，如果不是来自于本地的，比如说来自于网络上某处的与本地编码方式不同的机器，那么我们在构造输入流时就不能简单地使用本地缺省的编码方式，否则读出的字符就不正确；为了正确地读出异种机上的字符，我们应该使用下述方式构造输入流对象：

```
ir = new InputStreamReader(is, encode);
```

采用”encode”所指示的编码方式来构造输入字符流，编码方式有很多种，如：“ISO8859-1”，“GB2312”等。

输入/输出处理---URL对象

```
1 InputStream is = null;
2 String fileName = new String("Data/data.1-96");
3 byte buffer[] = new byte[24];
4
5 try {
6     // new URL throws a MalformedURLException
7     URL fileLocation = new URL(getDocumentBase(), fileName);
8
9     // URL.openStream() throws an IOException
10    is = fileLocation.openStream();
11 } catch (Exception e) {
12     // ignore
13 }
14 try {
15    is.read(buffer, 0, buffer.length);
16 } catch (IOException e1) {
17     // ignore
18 }
```

输入/输出处理---对象的串行化

- 对象记录自己的状态以便将来再生的能力，叫作对象的持续性(persistence)。
- 对象通过写出描述自己状态的数值来记录自己，这个过程叫对象的串行化(Serialization)。
- 在java.io包中，接口Serializable用来作为实现对象串行化的工具，只有实现了Serializable的类的对象才可以被串行化。

输入/输出处理---对象的串行化

```
public class Student implements Serializable{  
    int id;          //学号  
    String name;     //姓名  
    int age;         //年龄  
    String department //系别  
    public Student(int id, String name, int age, String department){  
        this.id = id;  
        this.name = name;  
        this.age = age;  
        this.department = department;  
    }  
}
```

输入/输出处理---对象的串行化

- java.io包中，提供了ObjectInputStream和ObjectOutputStream将数据流功能扩展至可读写对象。在ObjectInputStream中用readObject()方法可以直接读取一个对象，ObjectOutputStream中用writeObject()方法可以直接将对象保存到输出流中。

对象的 串行保存

```
1 import java.io.*;
2 import java.util.Date;
3
4 public class SerializeDate {
5
6     SerializeDate() {
7         Date d = new Date ();
8
9         try {
10             FileOutputStream f =
11                 new FileOutputStream ("date.ser");
12             ObjectOutputStream s =
13                 new ObjectOutputStream (f);
14             s.writeObject (d);
15             s.close ();
16         } catch (IOException e) {
17             e.printStackTrace ();
18         }
19     }
20
21     public static void main (String args[]) {
22         new SerializeDate();
23     }
24 }
```

对象的 串行读取

```
1 import java.io.*;
2 import java.util.Date;
3
4 public class UnSerializeDate {
5
6     UnSerializeDate () {
7         Date d = null;
8
9         try {
10             FileInputStream f =
11                 new FileInputStream ("date.ser");
12             ObjectInputStream s =
13                 new ObjectInputStream (f);
14             d = (Date) s.readObject ();
15             s.close ();
16         } catch (Exception e) {
17             e.printStackTrace ();
18         }
19
20         System.out.println (
21             "Unserialized Date object from date.ser");
22         System.out.println ("Date: "+d);
23     }
24
25     public static void main (String args[]) {
26         new UnSerializeDate();
27     }
28 }
```

输入/输出处理---对象的串行化

- 串行化能保存的元素

只能保存对象的非静态成员变量，不能保存任何的成员方法和静态的成员变量，而且串行化保存的只是变量的值，对于变量的任何修饰符，都不能保存。

- transient关键字

对于某些类型的对象，其状态是瞬时的，这样的对象是无法保存其状态的，例如一个Thread对象，或一个FileInputStream对象，对于这些字段，我们必须用transient关键字标明。