Java软件设计基础

中国科学技术大学计算机科学与技术学院

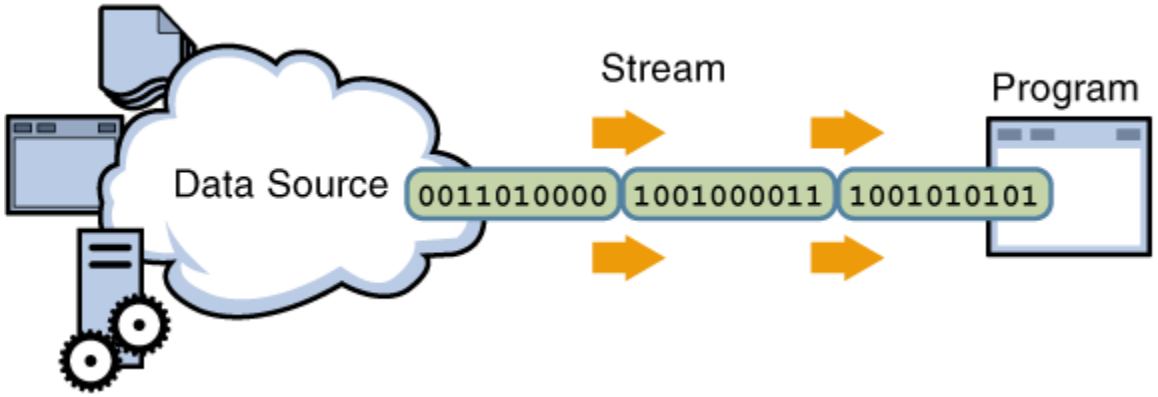
8. 输入输出流

1. 流的概念

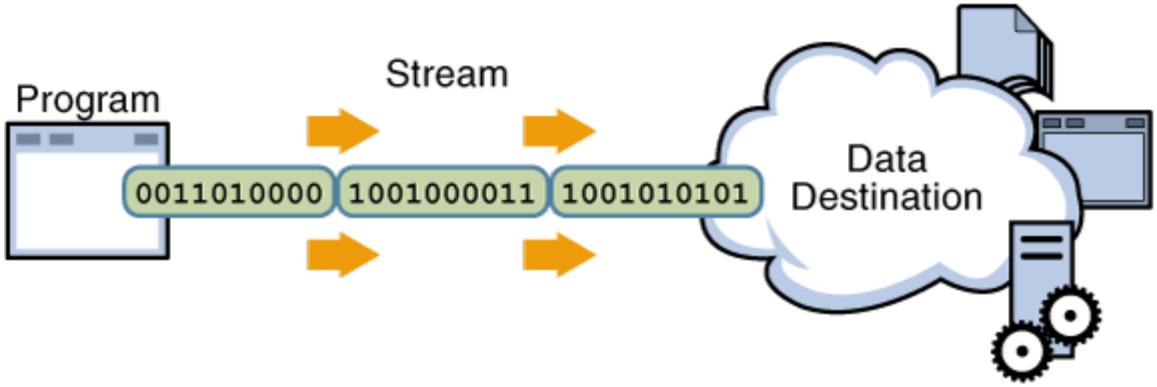
- 流(stream)
  - 流是指按照顺序组织的、从起点到终点的数据的集合。
  - 流的特点是数据的发送与获取都是按数据序列顺序进行的，每个数据必须等待它前面的数据发送或读入才能被读写。
  - 流式输入/输出是一种很常见的输入/输出方式，它是一个比文件所包含的范围更为广泛的概念。
  - Java提供了丰富的输入/输出操作，它们都是从“流”读出和向“流”写入的。任何Java中表示数据源的对象都会提供以数据流的方式读写它的数据的方法。
  - I/O流表示一个输入来源或者输出目标，支持很多不同类型的数据，如简单字节、基本数据类型、本地化字符和对象。

- 流的分类
  - 按照流向可分为输入流和输出流。
    - 数据流向是从当前Java程序运行所在的角度来划分的：
      - 输入流：将数据导入到程序中；
      - 输出流：程序将数据导出到到外部设备。
  - 根据操作数据的单位，可分为字符流和字节流。
    - 字节流：由InputStream和OutputStream作为基类
    - 字符流：由Reader和Writer作为基类
  - 按照流的角色来分，可分为节点流（低级流）和处理流（包装流）。
    - 节点流：从/向一个IO设备即实际的数据源进行读/写数据的流
    - 处理流：对一个已经存在的流进行连接或包装，通过包装后的流实现数据读写功能

- Java语言将流的相关操作分为两个部分：
  - 标准输入/输出流：包含在java.lang中，用于标准输入/输出设备的流操作；
  - 更丰富的流操作：包含在java.io中，要使用这些流类必须先用import语句引入。
- 流的工作流程
  - 读入数据：
    - 程序先从数据源(data source)打开一个流，然后从这个流中顺序读取数据，最后关闭流。



- 输出数据
  - 程序打开一个流，通过这个流中输出目标顺向数据宿(data destination)写入数据，最后关闭流。



2. 输入流和输出流的读写

- InputStream和Reader
  - 是所有输入流的抽象父类，不能创建实例来执行具体的输入，是所有输入流的模板。
  - 读取数据的方法

int read( )

从输入流中读取单个字节/字符，返回所读取的字节/字符数据（字节、字符数据可以直接转换为int类型）

流是通过-1来标记结束的。

在用输入流读取一个字节（byte）数据时，有时会出现**连续8个1**的情况，这个值在计算机内部表示-1，正好符合了流结束标记。为了避免流操作数据提前结束，将读到的字节进行int类型的无符号扩展。保留该字节数据的同时，前面都补0，避免出现-1的情况。（char同理）

int read(byte buff[ ])
int read(char cbuff[ ])

从输入流中最多读取buff.length/cbuff.length个字节/字符的数据，并将其存放在字节/字符数组buff/cbuff中，返回实际读取的字节/字符数

int read(byte buff[ ],int start,int len)
int read(char cbuff[ ], int start, int len)

从输入流中最多读取len个字节/字符的数据，并将其存放在字节/字符数组buff/cbuff中，起点从start位置开始，返回实际读取的字节/字符数

- 其他方法

void close( )

- 关闭当前流对象，回收释放此连接所占用的资源；程序中打开的IO资源不属于内存里的资源，垃圾回收机制无法回收该资源，所以应显式关闭。
- Java 7改写了所有的IO资源类，实现了AutoCloseable接口，可通过自动关闭资源的try语句来关闭这些IO流。

void mark( )

- 在记录指针当前位置记录一个标记

boolean markSupport( )

- 判断此输入流是否支持mark操作，即是否支持记录标记

void reset( )

- 将该流的记录指针定位到上一次记录标记（mark）的位置

long skip(long n)

- 记录指针向前移动n个字符

- OutputStream和Writer
  - 是所有输出流的抽象父类，同样的，也不能创建该类的实例用于输出。
  - 输出数据的方法

```
void write(int c)
```

    - 将指定的字节/字符输出到输出流中

```
void write(byte buff[ ])
void write(char cbuff[ ])
```

    - 将字节/字符数组中的数据输出到指定流中

```
void write(byte buff[ ], int off, int len)
void write(char cbuff[ ], int off, int len)
```

    - 将字节/字符数组中的数据从off开始，输出len长度的字节/字符到指定流中

void write(String str)

- 将str字符串里包含的字符输出到指定的字符输出流中

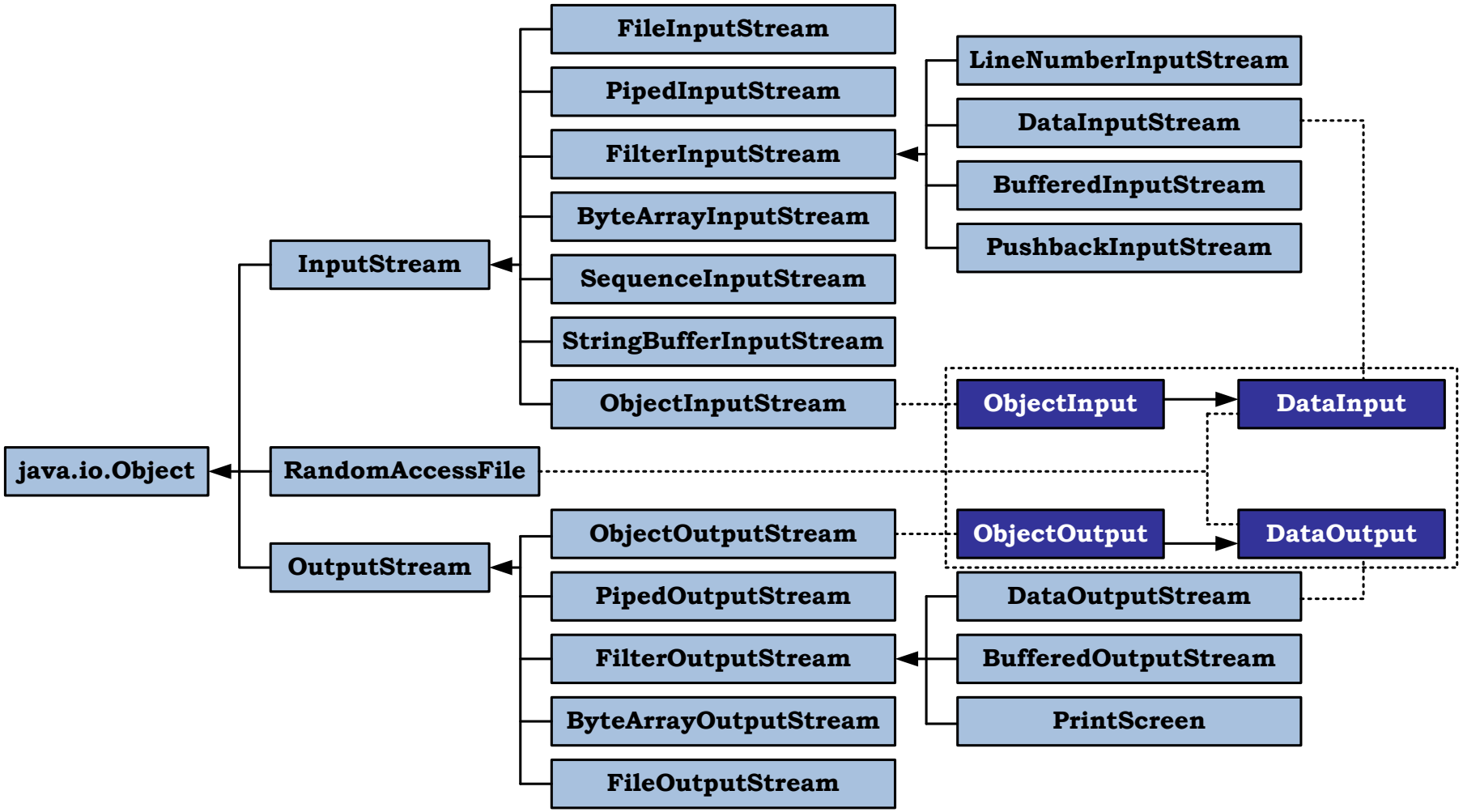
void write(String str, int off, int len)

- 将str字符串里从off开始、长度为len的字符输出到指定的字符输出流中

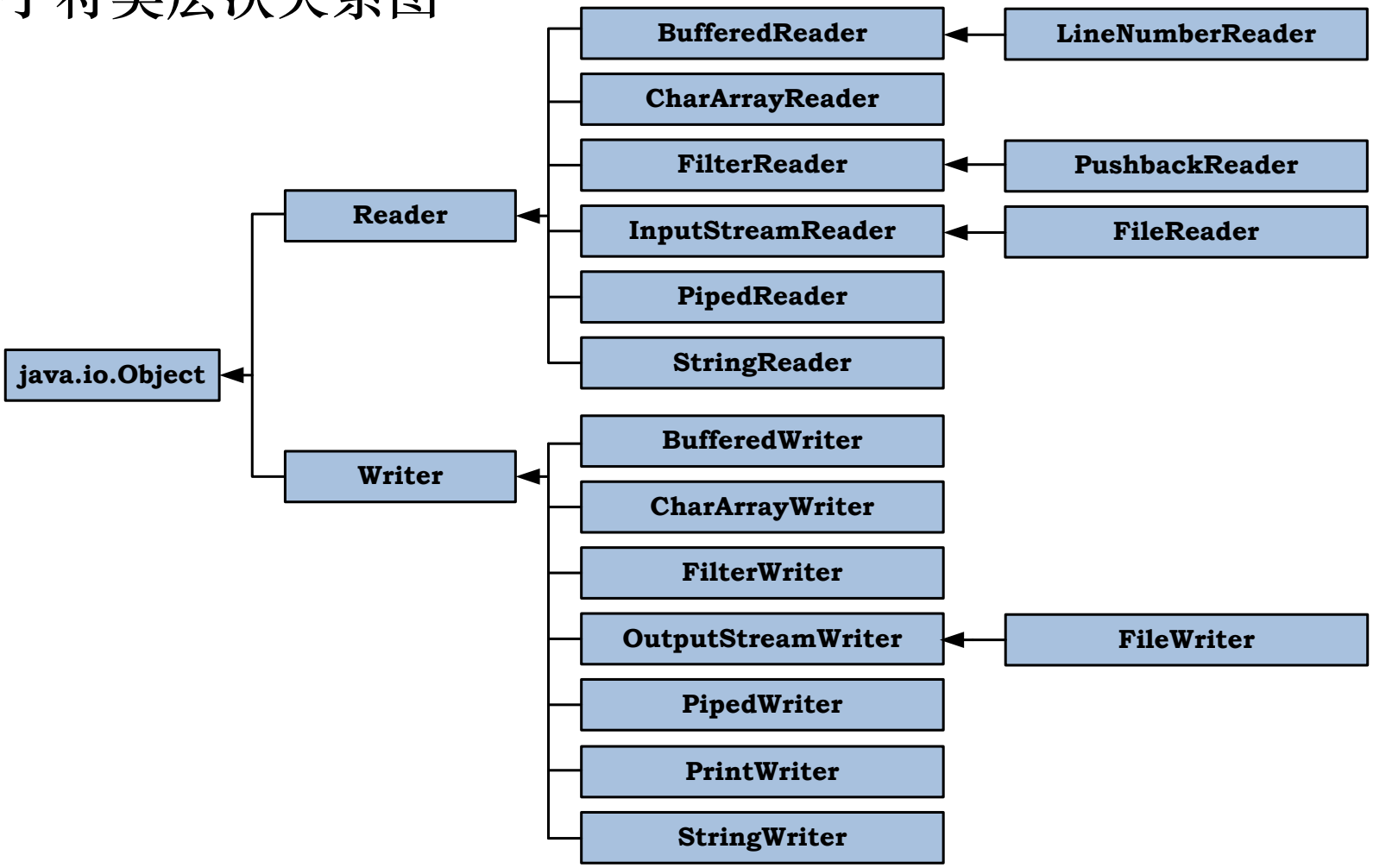
关闭输出流除了可以保证流的物理资源被回收之外，可能还可以将输出流缓冲区中的数据flush到物理节点里（因为在执行close()方法之前会自动执行输出流的flush()方法）。

3. 输入/输出流体系

- 字节流类层次关系图



- 字符类层次关系图


java.io.Object
Reader
BufferedReader
LineNumberReader
CharArrayReader
FilterReader
PushbackReader
InputStreamReader
FileReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
FileWriter
PipedWriter
PrintWriter
StringWriter

分类	字节输入流	字节输出流	字符输入流	字符输出流
抽象流	*InputStream*	*OutputStream*	*Reader*	*Writer*
节点流	FileInputStream	FileOutputStream	FileReader	FileWriter
	ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
	PipedInputStream	PipedOutputStream	PipedReader	PipedWriter
			StringReader	StringWriter
处理流	BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
			InputStreamReader	InputerStreamWriter
	ObjectInputStream	ObjectOutputStream		
抽象流	*FilterInputStream*	*FilterOutputStream*	*FilterReader*	*FilterWriter*
处理流		PrintStream		PrintWriter
	PushbackInputStream		PushbackReader	
	DataInputStream	DataOutputStream		

System.in

- 定义在System类中的一个标准输入流对象：

  public static final InputStream in

- 此流已打开并准备提供输入数据。通常，此流对应于键盘输入或者由主机环境或用户指定的另一个输入源。

```
import java.io.*;
public class InputClass{
    public static void main(String args[]) throws IOException{
        byte buff[]=new byte[5];
        System.in.read(buff);
        String str=new String(buff,0);
        System.out.println(str);
    }
}
```

将一个字节型数组转换成字符串数组，参0表示将字符串数组中每个元素的高8位置为0

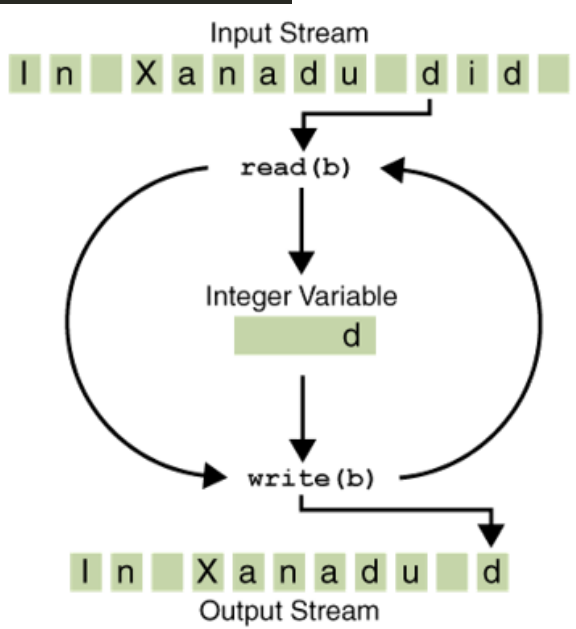
数组流

- 例：ByteArrayInputStream和ByteArrayOutputStream

```
import java.io.*;
public class RWByteArray{
    public static void main(String args[])throws IOException{
        byte[] b1=new byte[]{'a','r','r','a','y','1'};
        byte[] b2=new byte[64];
        ByteArrayInputStream in=new ByteArrayInputStream(b1);
        ByteArrayOutputStream out=new ByteArrayOutputStream();
        int c;
        while((c=in.read())!=-1){
            out.write(c);
        }
        b2=out.toByteArray();
        if(in!=null)in.close();
        if(out!=null)out.close();
        for(int i=0;i<b2.length;i++)
            System.out.print((char)b2[i]);
    }
}
```

文件流

```java
import java.io.*;
public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        FileReader fr = null;
        FileWriter fw = null;
        try {
            fr = new FileReader("xanadu.txt");
            fw = new FileWriter("characteroutput.txt");
            int c;
            while ((c = fr.read()) != -1) {
                fw.write(c);
            }
        } finally {
            if (fr != null)
                fr.close();
            if (fw != null) {
                fw.close();
            }
        }
    }
}
```



转换流

- 用于实现将字节流转换为字符流

```
import java.io.*;
public class KeyinTest {
  public static void main(String[] args)  {
      try(
          InputStreamReader isr=new InputStreamReader(System.in);
          BufferedReader br=new BufferedReader(isr)) {
              String line=null;
              while((line=br.readLine())!=null){
                  if(line.equals("exit")){
                      System.exit(1);
                  }
                  System.out.println("input:"+line);
              }
          }catch(IOException ie){
              ie.printStackTrace();
      }
  }
}
```

缓冲流

- 非缓冲(unbuffered)I/O中，每个读取和写入请求都是由底层操作系统直接处理的，这使得程序的效率非常低，因为每个读写请求都会触发磁盘访问、网络活动或者其他一些代价相当高的操作。
- 为了降低以上开销，Java提供缓冲(buffered)I/O流。这是一种常见的性能优化方法，可以从缓冲流中成批的读取/写入字符，而不会每次都引起直接对数据源/数据目标的读/写操作。

- 原理：
  - 缓冲输入流从一个内存区域(称为缓冲区)读取数据，只当缓冲区为空时才调用本机API；缓冲输出流将数据写入缓冲区，只能缓冲区为满时才调用本机API。
  - 一般缓冲区的大小为内存页或磁盘块的整数倍。

- Java中有四个用来包装缓冲流的类
  - BufferedInputStream/BufferedOutputStream
  - BufferedReader/BufferedWriter
- 通过将非缓冲流对象传递给缓冲流类的构造方法，将非缓冲流“包装”成缓冲流。

```
import java.io.*;
public class FileReaderDemo{
    public static void main(String args[])throws Exception{
        BufferedReader br=new BufferedReader(new FileReader("FileReaderDemo.java"));
        String s;
        while((s=br.readLine())!=null){
            System.out.println(s);
        }
        br.close();
    }
}
```

回溯流

- 回溯流提供一个退回缓冲区，提供的unread方法可将读取出来的内容退回到该缓冲区。
- 当该回溯流再次调用read方法时，总是先从退回缓冲区读取，只有完全读取了退回缓冲区的内容之后才继续从原输入流中读取。
- 默认退回缓冲区的长度为1，在创建该回溯流对象时可指定该缓冲区的大小。
- 如果程序中退回到该缓冲区中的内容超出了其大小，会引发Pushback buffer overflow的IOException。

回溯流例程

```
import java.io.*;
public class PushBackReaderDemo{
        public static void main(String args[]) throws IOException{
                String s="if(a==4)a=0;\n";
                char buff[]=new char[s.length()];
                s.getChars(0,s.length(),buff,0);
                CharArrayReader in=new CharArrayReader(buff);
                PushbackReader f=new PushbackReader(in);
                int c;
                while((c=f.read())!=-1){
                        switch(c){
                                case '=':
                                        if((c=f.read())=='=')
                                                System.out.print("等于");
                                        else{
                                                System.out.print("赋值为");
                                                f.unread(c);
                                        }
                                        break;
                                default:
                                        System.out.print((char)c);
                                        break;
                        }
                }
        }
}
```

运行结果

```
if(a等于4)a赋值为0;
```

4. 文件处理

- File类
  - File类提供了描述文件和目录的一种方法，专门用来管理磁盘文件和目录。
    - 目录可被视为一种特殊的文件；
    - 文件是保存在磁盘等存储设备上的数据，由记录组成，文件的一行可看作是一条记录；
    - 可以使用File类创建的对象来获取文件本身的一些信息。
  - File类在java.io包中，但不是流(InputStream或OutputStream)的子类，不负责数据的输入/输出；
    - 文件对象并不涉及对文件的读写操作；
    - 文件的读写要采取专门对文件操作的流，并应该在合适的时候关闭流，否则系统资源无法得到释放。

- File类的方法
  - 构造方法
    - public File(String pfname)
      - pfname指明了新建File对象所对应的磁盘文件或目录名及其路径名；路径包括相对路径与绝对路径，例如：

```
C:\\a1\\Example.java
A1/Example.java
```

      - 通常为了保证程序的可移植性，使用相对路径较好。
      - 在Java中，反斜杠“\”是一个特殊的字符，应该写成\\的形式。斜杠“/”是Java中目录的分隔符。

- public File(String dirPath, String fileName)
  - 参数dirPath表示对应文件或目录的路径名，fileName是不带路径名的文件名，例如：

```
File f = new File("docs","file.dat");
```

- public File(File f, String fileName)
  - f是文件所在目录的文件对象，fileName是不带路径名的文件名，例如：

```
File fDir = new File("a1/java");
File fFile = new File(fDir,"file.dat");
```

- 成员方法
  - 获取文件常规信息

public long length()	返回文件长度
public long lastModified()	返回文件最后修改日期

- 文件名相关方法

public String getName()	获取一个文件的文件名
public String getPath()	获取File对象对应的路径名
public File getAbsoluteFile()	获取File对象的绝对路径
public String getAbsolutePath()	获取File对象对应的绝对路径名
public String getParent()	返回File对象对应目录的父目录
boolean renameTo(File newName)	重命名文件

- 文件检测相关方法

public boolean exists()	判断文件是否存在
public boolean canWrite()	判断文件是否可写
public boolean canRead()	判断文件是否可读
public boolean isFile()	判断对象是否为文件
public boolean isDirectory()	判断对象是否为目录
public boolean isAbsolute()	判断文件对象对应的文件/目录是否为绝对路径

Microsoft Expression

- 文件操作方法

public boolean delete( )	删除文件对象对应的文件/目录
boolean createNewFile()	当该File对象对应的文件不存在时新建一个文件
static File createTempFile(String prefix,String suffix) static File createTempFile(String prefix,String suffix,File dir)	在默认目录或给定目录中创建一个临时空文件，具有指定前缀、随机数、指定后缀名
void deleteOnExit()	注册一个删除钩子当JVM退出时删除该File对象对应的文件/目录

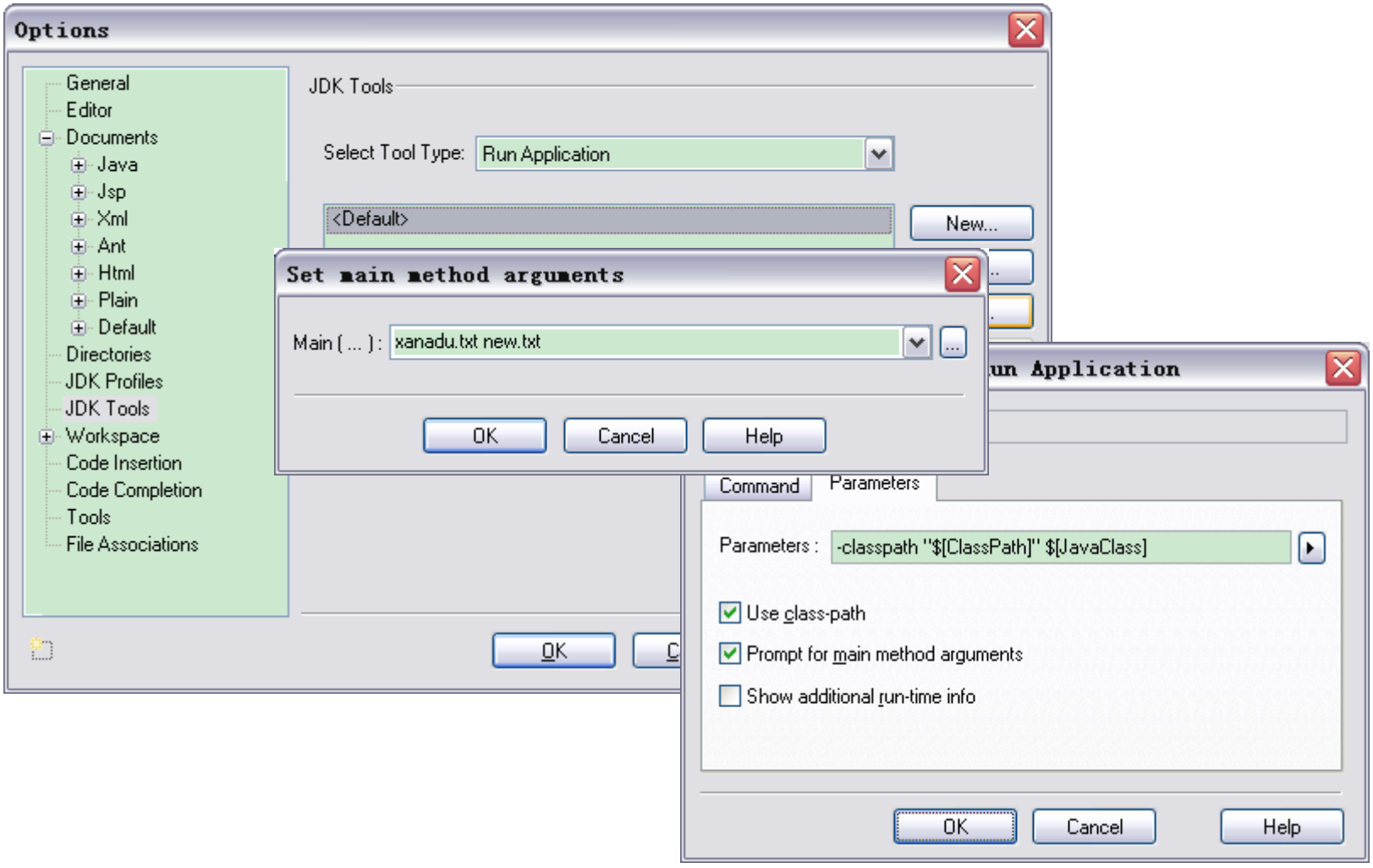
- 目录操作方法

public boolean mkdir( )	根据当前对象创建目录
public String[ ] list( )	列出当前目录下的文件
public File[ ] listFile( )	返回目录中所有文件对象列表
static File[ ] listRoots()	列出系统所有的根路径

Microsoft Expression

缓冲流例程

```
import java.io.*;
public class CopyChar{
        public static void main(String args[]) throws IOException{
                String sFile,dFile;
                if(args.length<2){
                        System.out.println("USE:java CopyChar 源文件名 目标文件名");
                        return;
                }
                else{
                        sFile=args[0];
                        dFile=args[1];
                }
                try{
                        File inputFile=new File(sFile);
                        File outputFile=new File(dFile);
                        FileReader in=new FileReader(inputFile);
                        BufferedReader bin=new BufferedReader(in);
                        FileWriter out=new FileWriter(outputFile);
                        BufferedWriter bout=new BufferedWriter(out);
                        String s;
                        while((s=bin.readLine())!=null){
                                bout.write(s);
                                bout.write("\n");
                        }
                        bin.close();
                        bout.close();
                }catch(IOException e){System.out.println(e.toString());}
        }
}
```


Options
General
Editor
Documents
Java
Jsp
Xml
Ant
Html
Plain
Default
Directories
JDK Profiles
JDK Tools
Workspace
Code Insertion
Code Completion
Tools
File Associations
JDK Tools
Select Tool Type:
Run Application
<Default>
New...
OK
Set main method arguments
Main (...) :
xanadu.txt new.txt
OK
Cancel
Help
un Application
Command
Parameters
Parameters :
-classpath "$[ClassPath]" $[JavaClass]
Use class-path
Prompt for main method arguments
Show additional run-time info
OK
Cancel
Help

- 获取目录及文件信息

```
public static void getInfo(File f1)throws IOException{
        SimpleDateFormat sdf;
        sdf=new SimpleDateFormat("yyyy年MM月dd日hh时mm分");
        if(f1.isFile())
                System.out.println("<FILE>\t"+f1.getAbsolutePath()+"\t"+
                f1.length()+"\t"+sdf.format(new Date(f1.lastModified())));
        else{
                System.out.println("\t"+f1.getAbsolutePath());
                File[] files=f1.listFiles();
                for(int i=0;i<files.length;i++)
                        getInfo(files[i]);
        }
}
```

SimpleDateFormat是一个以国别敏感的方式格式化和分析日期类数据的具体类。
它允许格式化、语法分析和标准化。
format方法将获取到的文件最后修改时间以格式sdf的方式表示。

当参数f1为文件类型，则打印<FILE>标签和文件的具体信息，如路径、大小、日期等；
当参数f1不是文件类型，则打印f1目录信息，并通过listFiles方法深入获取目录下的文件信息。
再通过递归调用方法层层深入，打印出该目录下所有的子目录、文件信息。

- 文件过滤器FileFilter和FilenameFilter
  - Java提供了文件过滤器(包括FileFilter和FilenameFilter两个接口)用来对文件名字符串进行筛选，以便获得满足需求特征的文件集合。
  - 以上两个接口都包含accept方法，实现accept方法就是制定自己的规则——哪些文件该由list()或listFiles()方法列出： ：

FileFilter

```
public interface FileFilter{
        public boolean accept(File pathname);
}
```

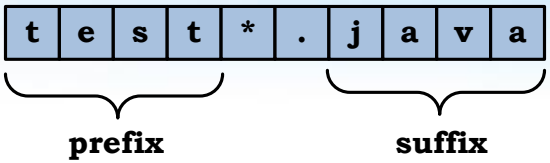
FilenameFilter

```
public interface FilenameFilter{
        public boolean accept(File pathname,String name);
}
```

  - 文件过滤器的实现类的实例可以作为File类的列表方法的参数，用于获得符合过滤要求的文件列表:

```
public String[] list(FilenameFilter filter)
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
```

- 例程：列出当前目录中带过滤器的文件名清单



- 在实现类DirFilter的构造方法中分析过滤器：

```
public DirFilter(String filterstr){
        String fs=filterstr.toLowerCase();
        int i=fs.indexOf('*');
        int j=fs.indexOf('.');
        if(i>0)
                prefix=fs.substring(0,i);
        if(j>0)
                suffix=fs.substring(j+1);
}
```

变为全小写

如果过滤器含有通配符*, 则获取*以前的字符

得到文件后缀名

- 判断文件是否满足过滤条件(即实现FilenameFilter中的accept方法)

```
public boolean accept(File dir,String filename){
        boolean yes=true;
        try{
                filename=filename.toLowerCase();
                yes=(filename.startsWith(prefix))&(filename.endsWith(suffix));
        }catch(NullPointerException e){
                System.out.println(e.toString());
        }
        return yes;
}
```

是否以prefix开头并且以suffix结尾

```
public static void main(String args[])throws IOException{
        FilenameFilter filter=new DirFilter("test*.java");
        File f1=new File("");
        File curdir=new File(f1.getAbsolutePath(),"");
        System.out.println(curdir.getAbsolutePath());
        String str[]=curdir.list(filter);
        for(int i=0;i<str.length;i++)
                System.out.println("\t"+str[i]);
}
```

创建过滤器对象"text*. java"

获得当前目录

在当前目录中筛选满足过滤器的文件，返回到数组str[]中。
在list方法的实现中，会自动根据accept方法的返回值决定是否将文件名加入到str[]中。

运行结果

```
E:\javadoc
    TestInner2.java
    test.java
    TestInner.java
```

5. 扫描和格式化

- 扫描Scanner
  - Java平台提供的把输入分割为和数据位相关联的记号的A
  - 把输入分割为记号
    - 默认情况下，Scanner使用空白分隔记号。

扫描例程

```
import java.io.*;
import java.util.Scanner;
public class ScanXan {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        try {
            s = new Scanner(new BufferedReader(new FileReader("xanadu.txt")));
            while (s.hasNext()) { System.out.println(s.next());}
        } finally {
            if (s != null) {s.close();}
        }
    }
}
```

xanadu.txt

```
In Xanadu did Kubla Khan
A stately pleasure-dome decree:
Where Alph, the sacred river, ran
Through caverns measureless to man
Down to a sunless sea.
```

输出结果

```
In
Xanadu
did
Kubla
Khan
A
stately
pleasure-dome
decree:
Where
Alph,
the
sacred
river,
ran
Through
caverns
measureless
to
man
Down
to
a
sunless
sea.
```

- 当Scanner对象的工作完成时，也要调用Scanner的close方法。尽管扫描器不是一个流，但是需要关闭它表示完成了对其底层流的处理。

- 转换各个记号
  - Scanner还支持所有原始类型(char除外)的记号；
  - 数字值可以使用千位分隔符，如果地区为US时，Scanner可以正确的读取表示整数值的字符串“32,757”。
  - 需要注意的是因为千位分隔符和小数点符号是地区特定的，当没有指明Scanner应该使用的地区时，可能无法得到正确的结果。
  - 例程：从文件中读取数据并相加，得出结果。

usnumbers.txt内的数据

```
8.5
32,767
3.14159
1,000,000.1
```

```
import java.io.*;
import java.util.*;
public class ScanSum {
    public static void main(String[] args) throws IOException {
        Scanner s = null;
        double sum = 0;
        try {
            s = new Scanner(new BufferedReader(new FileReader("usnumbers.txt")));
            s.useLocale(Locale.US);
            while (s.hasNext()) {
                if (s.hasNextDouble()) sum += s.nextDouble();
                else s.next();
            }
        }finally {s.close();}
        System.out.println(sum);
    }
}
```

指明Scanner应用的地区

while循环内的语句对文件内容自动分割，并转换记号（如千位分隔符、小数点）

- 实现格式化的流对象是PrintWriter或PrintStream的实例。
  - print和println方法：按照标准方式格式化单个值。
  - format方法：
    - 根据格式字符串(format string)格式化多个实参。
    - 格式字符串由嵌入了格式说明符(format specifier)的静态文本构成；
    - 所有的格式说明符都以%开头，以一个或两个字符转换结束，指定生成的格式化输出的种类。

格式化例程

```
public class Root2 {
    public static void main(String[] args) {
        int i = 2;
        double r = Math.sqrt(i);
        System.out.format("The square root of %d is %f.%n", i, r);
    }
}
```

d：把整数值格式化为小数值
f：把浮点值格式化为小数值
n：输出平台特定的行终止符

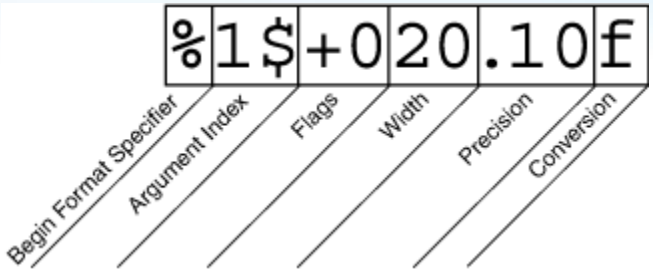
- 还有一些其他的转换，如：
  - **x：把整数格式化为16进制值；**
  - **s：把任何值格式化为字符串；**
  - **tB：把整数格式化为地区特定的月份名称**
  - **……**
- 除了%%和%n之外，所有格式说明符都必须和一个参数匹配，否则将抛出异常。

- 利用附加元素进一步定制格式化输出

```
public class Format {
    public static void main(String[] args) {
        System.out.format("%f, %1$+020.10f %n", Math.PI);
    }
}
```

输出

```
3.141593, +00000003.1415926536
```

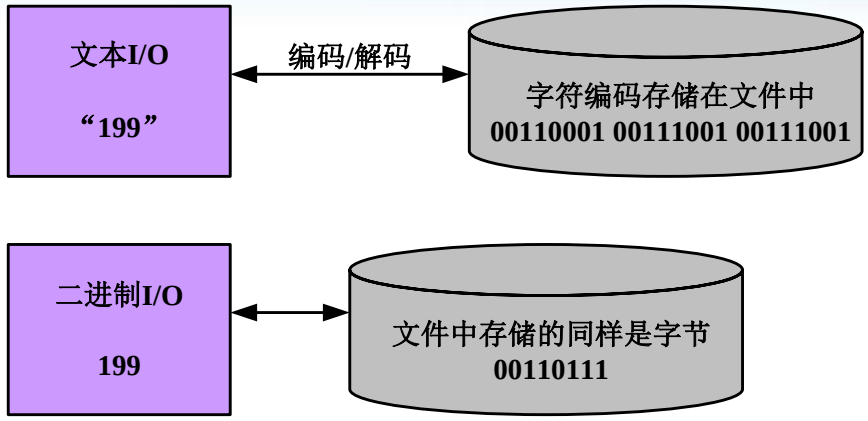


- 说明
  - 精度(Precision)：用于浮点值，是格式化值的数学精度。对于s和其他一般转换，它是格式化值的最大宽度；如果必要，对值进行右侧截断；
  - 宽度(Width)：格式化值的最小宽度；如果必要，就对值进行填补；
  - 标志(Flags)：指定附加的格式化选项。
    - **“+”标志指定总是把数字格式化为带有符号，“0”标志指定填补字符为0，“-”标志表示填补到右侧（默认是左侧），“,”表示使用千位分隔符等等。**
  - 实参索引(Argument Idex)：允许明确的匹配指定的实参，也可以使用“<”匹配和前一个说明符相同的实参。

```
System.out.format("%f, %2$+020.10f %n", Math.PI,Math.PI-1);
```

6. 数据流

计算机本身并不区分文本文件和二进制文件，本质上它们都是二进制格式来存储的。文本I/O是在二进制I/O的基础上提供的一层抽象，它封装字符的编码和解码过程。在写入和读取字符时自动进行编解码过程。



- 数据流支持基本数据类型值和String值的二进制I/O。
- 所有数据流实现DataInput接口或DataOutput接口。
- DataInputStream和DataOutputStream是以上两个接口的使用最广泛的实现，它们只能被创建为现有字节流对象的包装器（处理流）。

- 例程：通过数据流将一组数据写入到文件中，再从文件中读出数据。
  - 数据定义：

数据定义

```
static final String dataFile = "invoicedata";
static final double[] prices = { 19.99, 9.99, 15.99, 3.99, 4.99 };
static final int[] units = { 12, 8, 13, 29, 50 };
static final String[] descs = { "Java T-shirt",
                                "Java Mug",
                                "Duke Juggling Dolls",
                                "Java Pin",
                                "Java Key Chain" };
```

  - 打开一个输出流并写出数据记录

输出二进制流

```
DataOutputStream out = null;
try{
        out = new DataOutputStream(new FileOutputStream(dataFile));
        for (int i = 0; i < prices.length; i ++){
                out.writeDouble(prices[i]);
                out.writeInt(units[i]);
                out.writeUTF(descs[i]);
        }
}finally{
        out.close();
}
```

DataOutputStream只能被创建为现有字节流对象的包装器，
因此本程序中先建立FileOutputStream对象，再使用DataOutputStream进行“包装”；

writeDouble：写入双精度浮点数；
writeInt：写入整型；
writeUTF：按照UTF-8的一种修订形式写出String值，这是一种宽度可变的字符编码，
对于常用的西方字符，只需要一个字节。

- 数据流通过捕获EOFException异常检测文件结束条件，而不是通过检查返回值是否为-1来判断。
- 数据流中每个专门的write都严格的与专门的read相匹配，由程序员确保输出类型和输入类型按照这种方式匹配。
- 输入流由简单的二进制数据构成，没有办法表明各个值的类型或者它们从什么地方开始。

- 打开一个输入流读入数据：

输入二进制流

```
DataInputStream in = null;
double price,total = 0.0;
int unit;
String desc;
try{
        in = new DataInputStream(new FileInputStream(dataFile));
        while (true) {
                price = in.readDouble();
                unit = in.readInt();
                desc = in.readUTF();
                System.out.format("You ordered %d units of %s at $%.2f%n",unit, desc, price);
                total += unit * price;
        }
}catch(EOFException e){
        System.out.println(e.toString());
}
finally{
        System.out.format("For a TOTAL of: $%.2f%n",total);
        in.close();
}
```

- 在上述的程序中使用浮点数表示货币值，但是总的来说，浮点数并不适用于精确值，对于十进制小数尤其不适合。
  - 例如：0.1怎样用二进制精确的表示?
- 用于货币值的正确类型是java.math.BigDecimal，但它是对象类型，无法处理数据流，这时就需要引入对象流的概念。

- 对象序列化
  - 允许把Java对象转换为平台无关的二进制流，从而允许把这种二进制流持久的保存在磁盘上。
  - 如果需要让某个对象支持序列化机制，则必须让该类实现如下两个接口之一，Java很多类都已经实现了Serializable接口：

Serializable　　Externalizable

- 上面的程序实例改写为：
  - 修改数据

数据定义

```
static final BigDecimal[] prices = {
        new BigDecimal("19.99"),
        new BigDecimal("9.99");
        new BigDecimal("15.99");
        new BigDecimal("3.99");
        new BigDecimal("4.99");
};
```

  - 输入和输出的改动：

输出二进制流

```
ObjectOutputStream out = null;
try{
        out = new ObjectOutputStream(new FileOutputStream(dataFile));
        for (int i = 0; i < prices.length; i ++){
                out.writeObject(prices[i]);
                out.writeInt(units[i]);
                out.writeUTF(descs[i]);
        }
}finally{
        out.close();
}
```

- 同理，打开输入流读取数据时，将获取price的语句改为：

```
BigDecimal price;
price = (BigDecimal) in.readObject();
```

- 如果readObject没有返回预期的对象类型，试图把它转换为正确类型可能会抛出ClassNotFoundException。
- 运行结果：

```
You ordered 12 units of Java T-shirt at $19.99
You ordered 8 units of Java Mug at $9.99
You ordered 13 units of Duke Juggling Dolls at $15.99
You ordered 29 units of Java Pin at $3.99
You ordered 50 units of Java Key Chain at $4.99
售货记录读取完毕
For a TOTAL of: $892.88
```

- 自定义的类如果想实现序列化，则必须在类的声明中实现Serializable接口，该接口是一个标记接口，无需实现任何方法，只是表明该类是可以序列化的，否则将出现运行错误：

```
java.io.NotSerializableException: Rectangle
```

对象序列化

```
import java.io.*;
class Rectangle implements Serializable{
    double width,height;
    public Rectangle(double width,double height){
        this.height=height;
        this.width=width;
    }
}
public class WRRect{
    public static void main(String[] args){
        try(
            ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("rect.txt")))
        {
            Rectangle r1=new Rectangle(1.0,2.0);
            oos.writeObject(r1);
        }catch(IOException ie){
            ie.printStackTrace();
        }
    }
}
```

对象序列化

java.io.NotSerializableException: Point

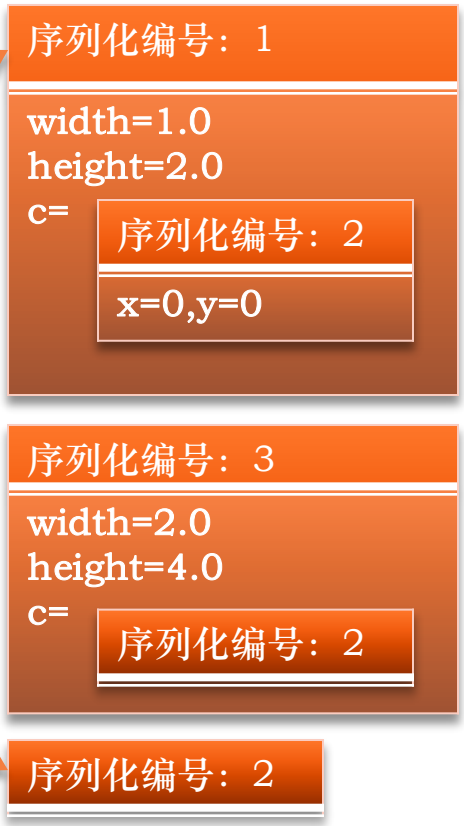
implements Serializable

```
class Point {
    double x,y;
    public Point(double x,double y){
        this.x=x;
        this.y=y;
    }
}
class Rectangle implements Serializable{
    double width,height;
    Point center;
    public Rectangle(double width,double height,Point center){
        this.height=height;
        this.width=width;
        this.center=center;
    }
}
```

当某个对象进行序列化时，系统会自动把该对象的所有实例变量依次进行序列化，如果实例变量引用到另一个对象(该类已经实现序列化)，则被引用对象也会被序列化；如果被引用的对象也有实例变量引用了其他对象……这种情况被称为递归序列化。

- Java的序列化机制
  - 所有保存到磁盘的对象都有一个序列化编号；
  - 当程序试图序列化对象时，先检查该对象是否已经序列号，只有当该对象从未序列化时系统才会将该对象转换为字节输出；
  - 如果某个对象已经被序列化，程序将只是直接输出一个序列化编号，而不是重新序列化该对象。

```
Point c=new Point(0,0);
Rectangle r1=new Rectangle(1.0,2.0,c);
Rectangle r2=new Rectangle(2.0,4.0,c);
oos.writeObject(r1);
oos.writeObject(r2);
oos.writeObject(c);
```



- 在序列化对象过程中，当某些实例变量信息不可序列化，或者因为某种原因不想序列化时：

[修饰符] transient 类型 实例变量名

- 将上例修改为：

对象序列化

```
class Rectangle implements Serializable{
    double width,height;
    Point center;
    transient String color="red";
    public Rectangle(double width,double height,Point center){
        this.height=height;
        this.width=width;
        this.center=center;
    }
}
```

transient只能修饰实例变量

```
public class WRRect{
    public static void main(String[] args){
        try(
            ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream("rect.txt"));
            ObjectInputStream ois=new ObjectInputStream(new FileInputStream("rect.txt")))
        {
            Point c=new Point(0,0);
            Rectangle r1=new Rectangle(1.0,2.0,c);
            oos.writeObject(r1);
            Rectangle r2=(Rectangle)ois.readObject();
            System.out.println(r2.color);
        }catch(IOException ie){
            ie.printStackTrace();
        }catch(ClassNotFoundException ce){
            ce.printStackTrace();
        }
    }
}
```

```
null
```

- 在序列化和反序列化过程中需要特殊处理的类，应该提供如下方法，用以实现自定义序列化：

```
private void writeObject(ObjectOutputStream oos) throws IOException
```

```
private void readObject(ObjectInputStream ois) throws
IOException,ClassNotFoundException
```

```
import java.io.*;
class Rectangle implements Serializable{
   ……
 private void writeObject(ObjectOutputStream oos)throws IOException{
       oos.writeDouble(height*10);
       oos.writeDouble(width*10);
   }
   private void readObject(ObjectInputStream ois)throws IOException,ClassNotFoundException{
       this.height=ois.readDouble(height/10);
       this.width=ois.readDouble(width/10);
   }
}
```

- 可以在序列化对象时将该对象替换成其他对象：

```
private Object writeReplace() throws ObjectStreamException
```

类Rectangle中实现以下方法：

```
private Object writeReplace() throws ObjectStreamException{
        ArrayList<Object> list=new ArrayList<Object>();
        list.add(height);
        list.add(width);
        return list;
}
```

读出对象时：

```
ArrayList list=(ArrayList)ois.readObject();
System.out.println("矩形对象"+list);
```

矩形对象[1.0, 2.0]

在使用writeObject之前**系统总是会自动调用**writeReplace方法，实际写入的并非Rectangle类的对象，而是ArrayList类的对象。
因此读取的时候读出的也是ArrayList对象。

- NIO
  - Java1.4开始提供了一系列新的输入/输出处理的功能以及新增类，采用将文件或文件的一段区域映射到内存中的方式。

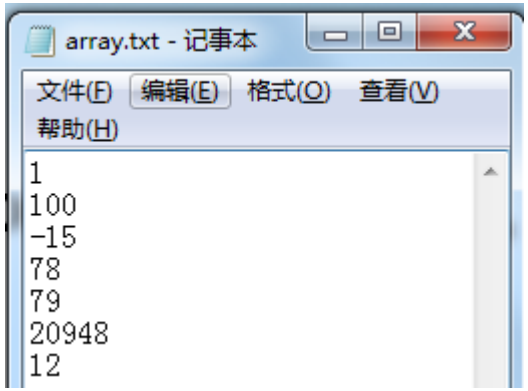
    如果说传统的IO是面向流的处理，则NIO是面向块的处理

  - Java7对NIO进行了改进，即NIO.2版本：
    - 提供了更丰富的文件IO和文件系统操作
    - 基于异步Channel的IO等

课堂练习

- 请将整型数组arrayA写到文件array.txt中。方法不限。
  - 如：

{1，100，-15，78，79，20948，12}



课下拓展（无需提交）：

压缩一个文本本件（假设该文件只由英文字母组成），你需要做的是：

1. 统计字母出现频次
2. 哈夫曼树的完成
3. 编码
4. 将编码代替字母写入压缩后的文件（怎么写？）

Microsoft Expression