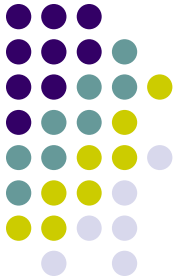
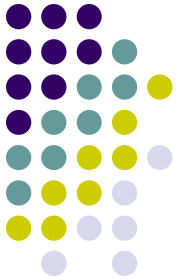
第六章 输入/输出流和文件



6.1 输入/输出流

6.2 文件读写

6.1 输入/输出流



- 通常程序需要从外部获取/输出信息
  - 这个“外部”范围很广，包括诸如键盘、显示器、文件、磁盘、网络、另外一个程序等
  - “信息”也可以是任何类型的，例如一个对象、串字符、图像、声音等
- 通过使用java.io包中的输入/输出流类就可以达到输入输出信息的目的

6.1.1 I/O流的概念

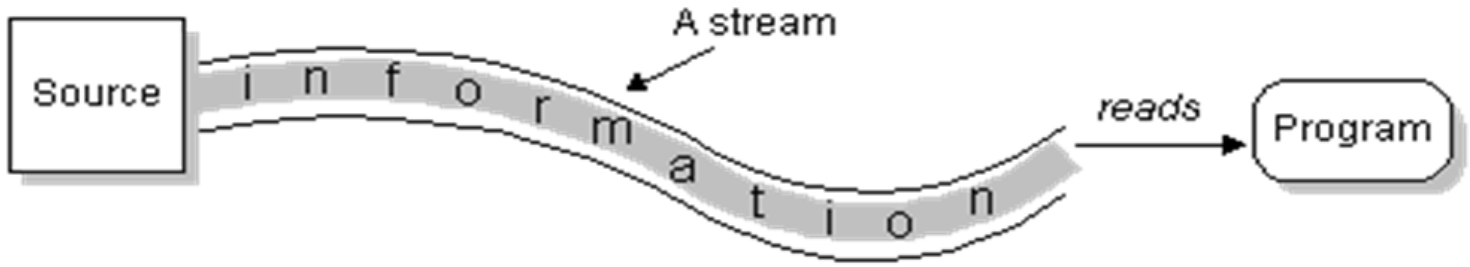
- 流：数据从计算机的输入向输出流动，即流的产生。
- 流有两种：文本流（字符）和二进制流（字节）
- 在Java里，流是一些类。
  - 在Java中将信息的输入与输出过程抽象为I/O流
    - 输入是指数据流入程序
    - 输出是指数据从程序流出
  - 一个流就是一个从源流向目的地的数据序列

6.1.1 I/O流的概念

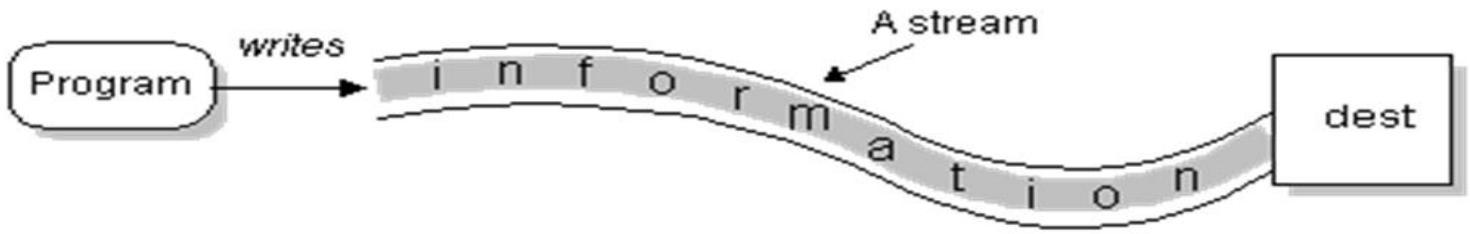
- 文件File也是一个逻辑概念。计算机的所有设备都可理解为一个文件。流可与文件建立联系。
- **java.lang.Object**
  - **java.io.File**
  - **java.io.RandomAccessFile**
  - **java.io.InputStream**
  - **java.io.OutputStream**
  - **java.io.Reader**
  - **java.io.Writer**
    - IO流类一旦被创建就会自动打开
    - 通过调用close方法，可以显式关闭任何一个流；
    - 如果流对象不再被引用，Java的垃圾回收机制也会隐式地关闭它

6.1.1 I/O流的概念

- 输入流
  - 为了从信息源获取信息，程序打开一个输入流，程序可从输入流读取信息



- 输出流
  - 当程序需要向目标位置写信息时，便需要打开一个输出流，程序通过输出流向这个目标位置写信息

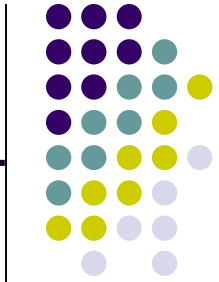


6.1.1 I/O流的概念

- 源和目标的类型

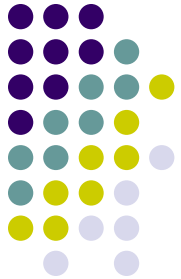
对象	源？目标？或两者？
disk file	Both
running program	Both
monitor	Destination
keyboard	Source
Internet connection	Both
image scanner	Source
mouse	Source

6.1.1 I/O流的概念



- 读写数据的方法
  - 不论数据从哪来，到哪去，也不论数据本身是何类型，读写数据的方法大体上都是一样的：
  - 打开一个流，读/写信息，关闭流。

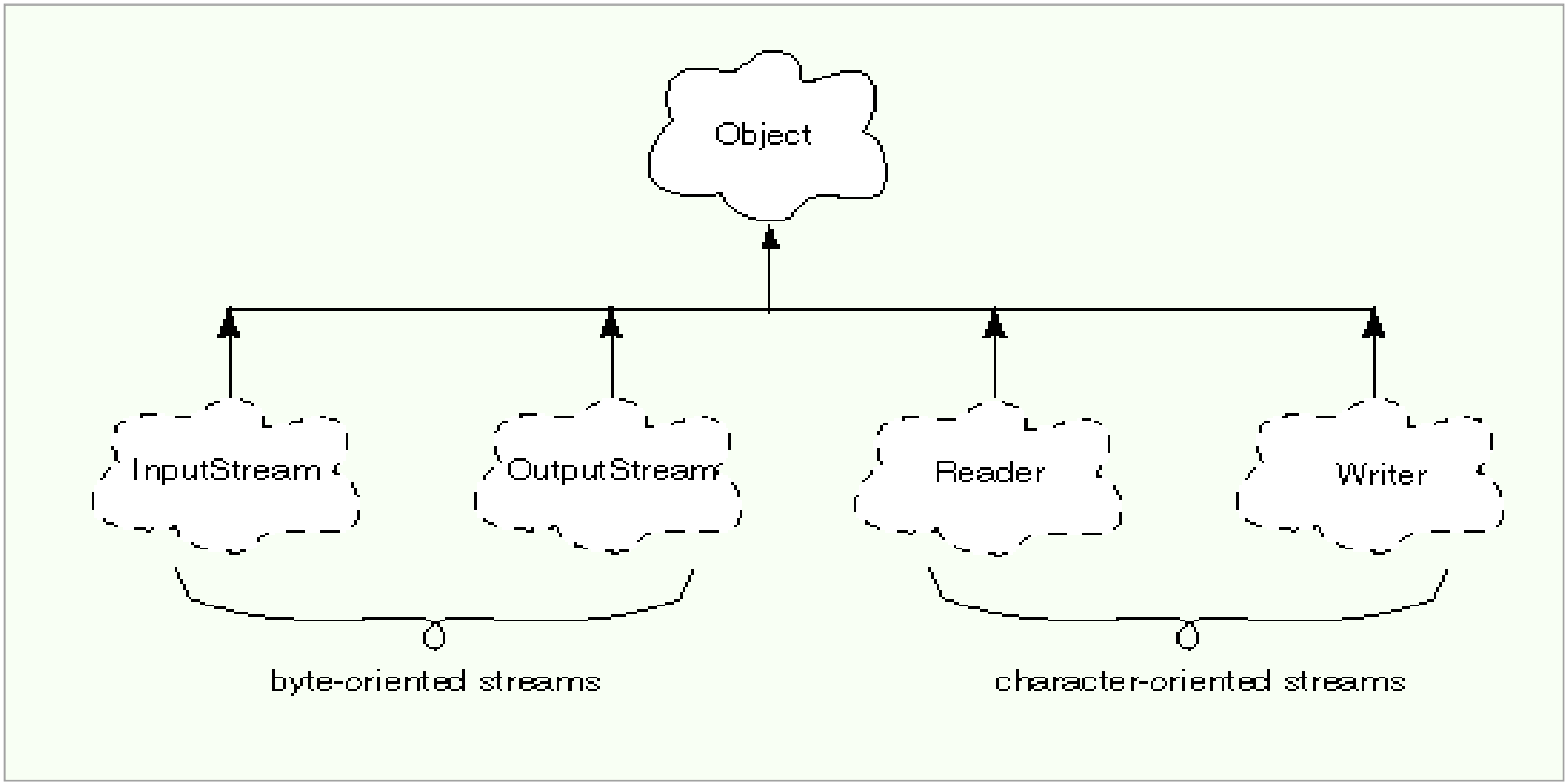
6.1.2 预定义的I/O流类概述



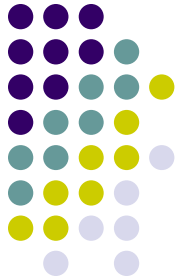
- 输入/输出流可以从以下几个方面进行分类
  - 从流的方向划分
    - 输入流
    - 输出流
  - 从流的分工划分
    - 节点流
    - 处理流
  - 从流的内容划分
    - 面向字符的流
    - 面向字节的流

6.1.2 预定义的I/O流类概述

- java.io包的顶级层次结构
  - 面向字符的流：专门用于字符数据
  - 面向字节的流：用于一般目的



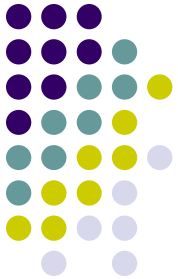
6.1.2 预定义的I/O流类概述



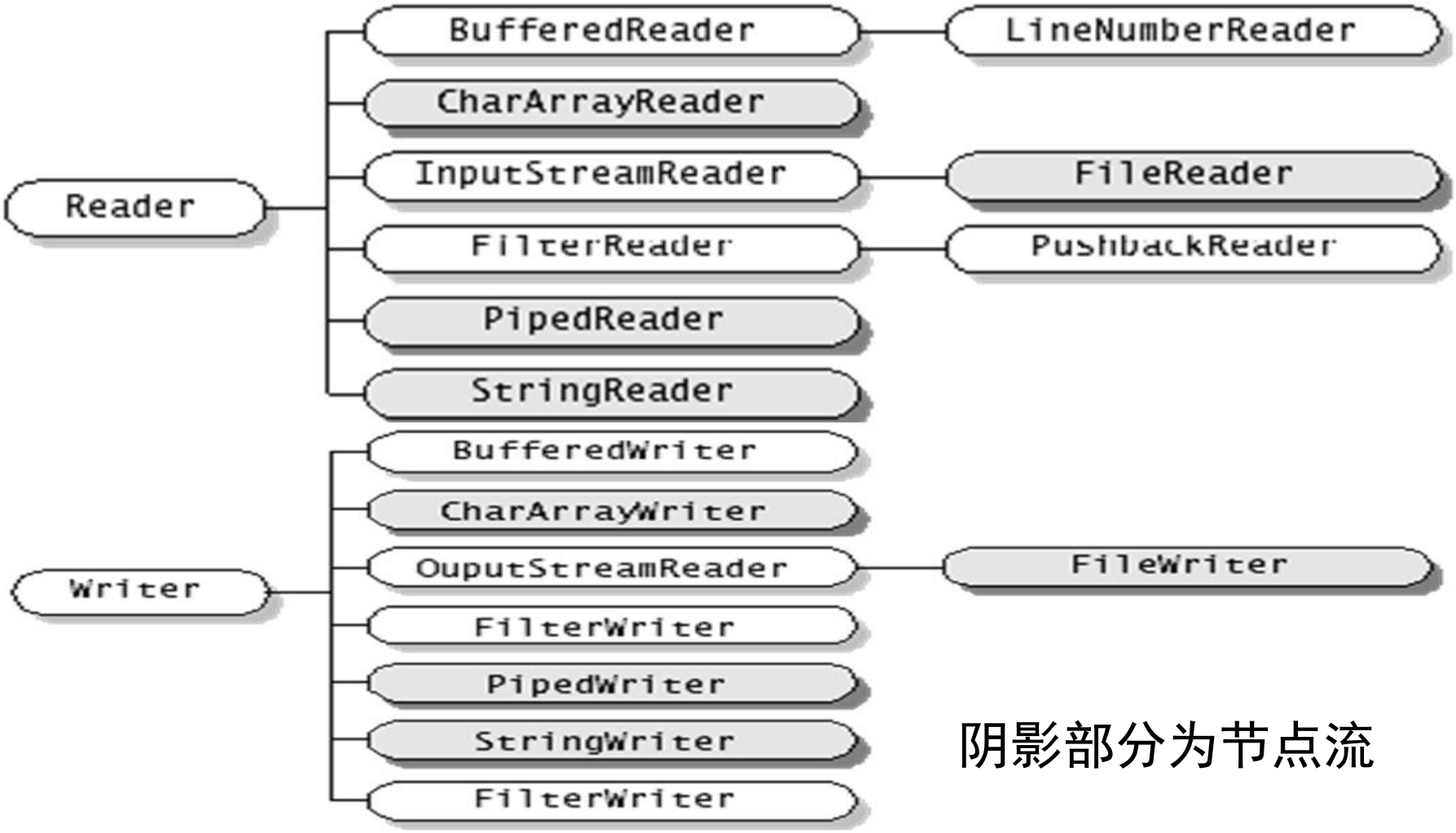
- 面向字符的流
  - 针对字符数据的特点进行过优化，提供一些面向字符的有用特性
  - 源或目标通常是文本文件

- 实现内部格式和文本文件中的外部格式之间转换
  - 内部格式：16-bit char 数据类型
  - 外部格式：
    - UTF(Universal character set Transformation Format)：很多人称之为"Universal Text Format"
    - 包括ASCII 码及非ASCII 码字符，比如： 斯拉夫(Cyrillic)字符, 希腊字符, 亚洲字符等

6.1.2 预定义的I/O流类概述

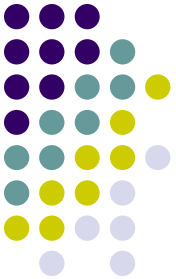


- 面向字符的抽象类——Reader和Writer
  - java.io包中所有字符流的抽象基类
  - Reader提供了输入字符的API
  - Writer提供了输出字符的API
  - 它们的子类又可分为两大类
    - 节点流：从数据源读入数据或往目的地写出数据
    - 处理流：对数据执行某种处理
  - 多数程序使用这两个抽象类的一系列子类来读入/写出文本信息
    - 例如FileReader/FileWriter用来读/写文本文件


Reader
BufferedReader
LineNumberReader
CharArrayReader
InputStreamReader
FileReader
FilterReader
PushbackReader
PipedReader
StringReader
Writer
BufferedWriter
CharArrayWriter
OuputStreamReader
FileWriter
FilterWriter
PipedWriter
StringWriter
FilterWriter

阴影部分为节点流

6.1.2 预定义的I/O流类概述



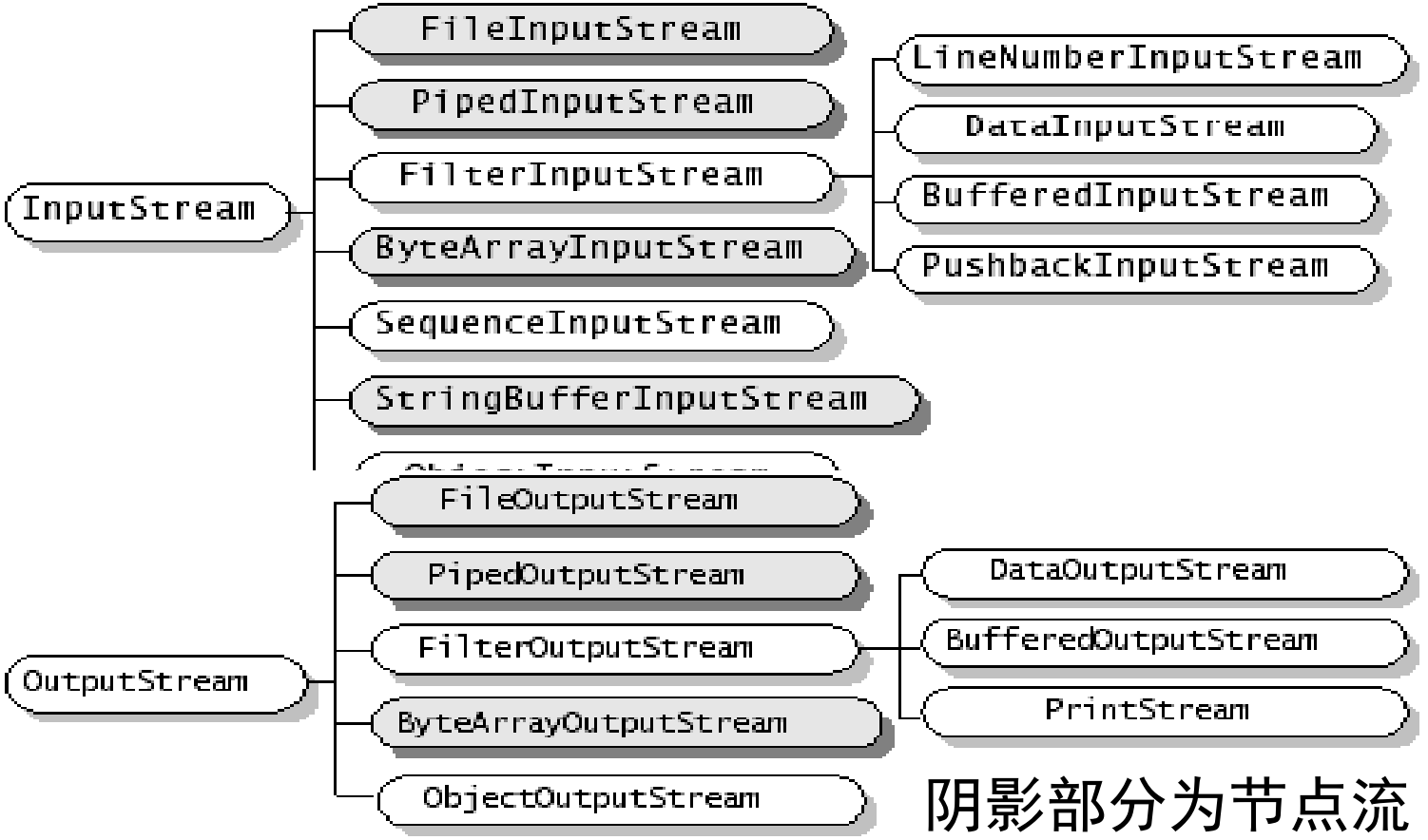
- 面向字节的流
  - 数据源或目标中含有非字符数据，必须用字节流来输入/输出
  - 通常被用来读写诸如图片、声音之类的二进制数据
  - 绝大多数数据是被存储为二进制文件的，世界上的文本文件大约只能占到2%，通常二进制文件要比含有相同数据量的文本文件小得多。

6.1.2 预定义的I/O流类概述

- 面向字节的流
  - InputStream和OutputStream
  - 是用来处理8位字节流的抽象基类，程序使用这两个类的子类来读写8位的字节信息
  - 分为两部分
    - 节点流
    - 处理流

6.1.2 预定义的I/O流类概述

- 面向字节的流



6.1.2 预定义的I/O流类概述

- 标准输入输出
  - 标准输入输出流属于System类。
  - System类实现了用户使用资源时的系统无关编程接口；是final类；所有变量和方法都是static的；不用初始化（new）就可以使用
  - System类的静态成员变量
    - System.in： InputStream类型的，代表标准输入流，这个流是已经打开了的，默认状态对应于键盘输入。
    - System.out：PrintStream类型的，代表标准输出流，默认状态对应于屏幕输出
    - System.err：PrintStream类型的，代表标准错误信息输出流，默认状态对应于屏幕输出

6.1.2 预定义的I/O流类概述

- 标准输入输出
  - System.in

    public final static InputStream in = new InputStream();

    read(), read(byte b[],int off, int len), read(byte b[])
  - System.out

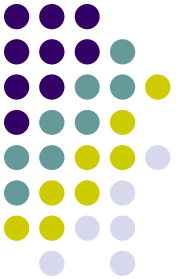
    public final static PrintStream out = new PrintStream();

    print(), println(), write()
  - System.err

    public final static PrintStream err = new PrintStream();

    print(), println(), write()

6.1.2 预定义的I/O流类概述



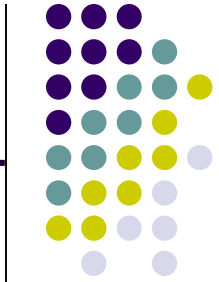
- 从键盘读入信息并在显示器上显示

```
import java.io.*;
public class Echo {
   public static void main(String[] args)  throws IOException {
      BufferedReader in = new BufferedReader(
                                 new InputStreamReader(System.in));
      String s;
      while((s = in.readLine()).length() != 0)
        System.out.println(s);
    }
 }
```

运行结果

```
Hello!
Hello!
```

6.1.2 预定义的I/O流类概述



- System.in
  - 程序启动时由Java系统自动创建的流对象，它是原始的字节流，不能直接从中读取字符，需要对其进行进一步的处理
- InputStreamReader(System.in)
  - 以System.in为参数创建一个InputStreamReader流对象，相当于字节流和字符流之间的一座桥梁，读取字节并将其转换为字符
- BufferedReader in
  - 对InputStreamReader处理后的信息进行缓冲，以提高效率

6.1.2 预定义的I/O流类概述

- 一个方便的扫描API:
  - 把文本转化成基本类型或者String

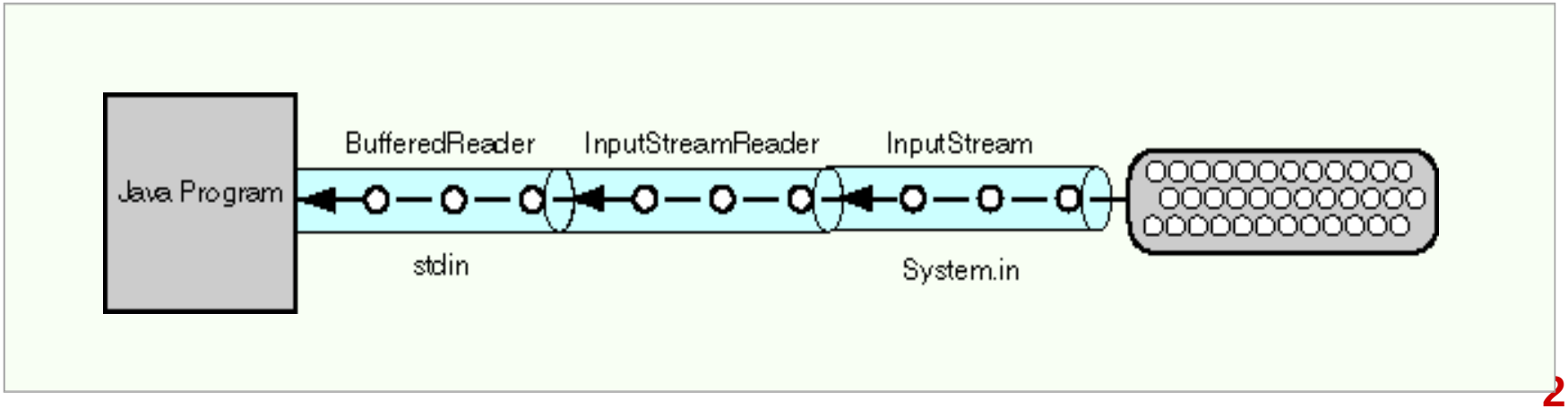
```
Scanner s = new Scanner(System.in);
int n = s.nextInt();
```

  - 还有下列方法：

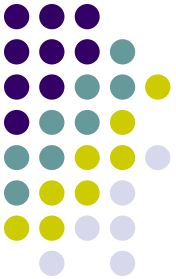
nextByte(),
nextDouble(),
nextFloat,
nextInt(),
nextLine(),
nextLong(),
nextShort()

6.1.2 预定义的I/O流类概述

- 处理流
  - 不直接与数据源或目标相连，而是基于另一个流来构造
  - 从流读写数据的同时对数据进行处理
  - InputStreamReader和BufferedReader都属于处理流
    - InputStreamReader读取字节并转换为字符
    - BufferedReader对另一个流产生的数据进行缓冲

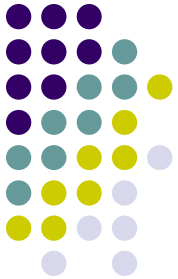


6.1.2 预定义的I/O流类概述



- I/O异常
  - 多数IO方法在遇到错误时会抛出异常，因此调用这些方法时必须
    - 在方法头声明抛出IOException异常
    - 或者在try块中执行IO，然后捕获IOException

第六章 输入/输出流和文件



6.1 输入/输出流

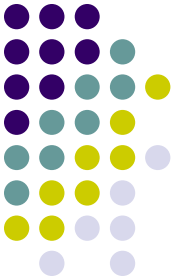
6.2 文件读写

- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.1 写文本文件

- FileWriter类
- 创建一个磁盘文件
- 关闭一个磁盘文件
- write() 方法
- 捕获I/O异常
- BufferedWriter 类

6.2.1 写文本文件



- 在C盘根目录创建文本文件Hello.txt，并往里写入若干行文本

```
import java.io.*;
class WriteTxtFile {
  public static void main (String[] args ) throws IOException {
     //main方法中声明抛出IO异常
     String fileName = “C:\Hello.txt";
     FileWriter writer = new FileWriter(fileName);
     writer.write( "Hello!\n");
     writer.write( "This is my first text file,\n"  );
     writer.write( "You can see how this is done.\n" );
     writer.write("输入一行中文也可以\n");
     writer.close();
  }
}
```

6.2.1 写文本文件

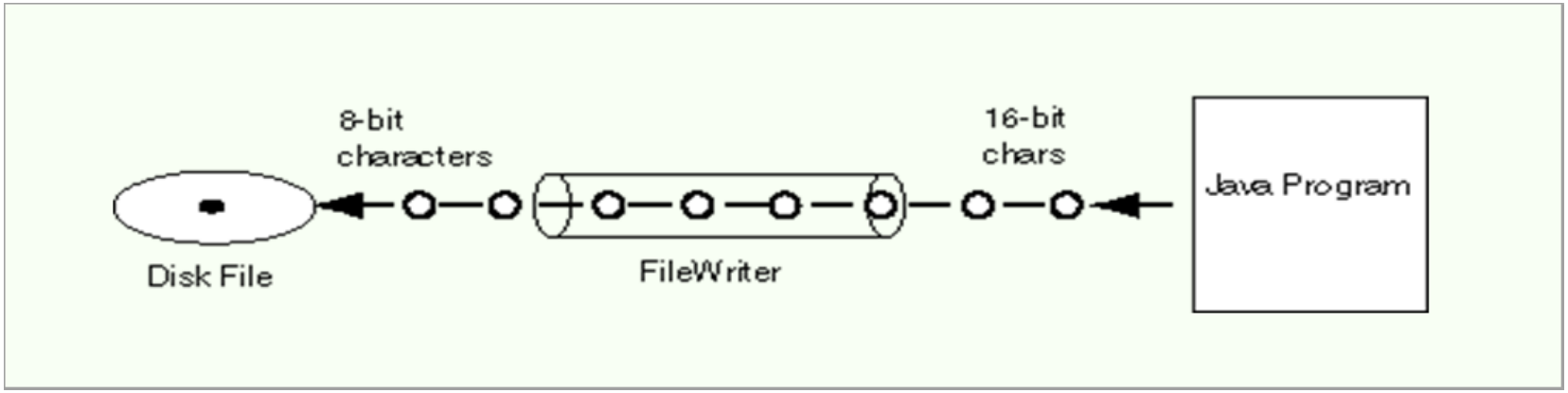
- 打开C盘根目录下的Hello.txt文件

  - 换行有些问题

- 每次运行这个程序，都将删除已经存在的”Hello.txt”文件，创建一个新的同名文件
- FileWriter的构造方法有五个，本例是通过一个字符串指定文件名来创建
- FileWriter类的write方法向文件中写入字符

6.2.1 写文本文件

- 上例WriteTxtFile说明
  - Writer类的流可实现内部格式到外部磁盘文件格式转换
  - “Hello.txt”是一个普通的ASCII码文本文件，每个英文字符占一个字节，中文字符占两个字节
  - Java程序中的字符串则是每个字符占两个字节的，采用Unicode编码
  - close方法清空流里的内容并关闭它。如果不调用该方法，可能系统还没有完成所有数据的写操作，程序就结束了

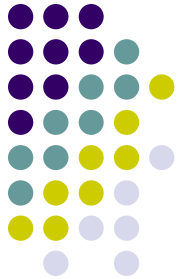


6.2.1 写文本文件

- 处理IO异常

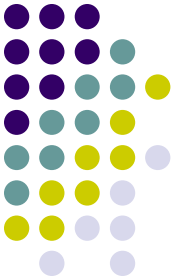
```
import java.io.*;
class WriteTxtFile2 {
  public static void main ( String[] args ) {
    String fileName = "c:\\Hello.txt" ;
    try {  //将所有IO操作放入try块中
                FileWriter writer = new FileWriter( fileName, true );
                writer.write( "Hello!\n");
                writer.write( "This is my first text file,\n"  );
                writer.write( "You can see how this is done. \n" );
                writer.write("输入一行中文也可以\n");
                writer.close();
           }
    catch ( IOException iox) {
       System.out.println("Problem writing" + fileName ); }
    }
}
```

6.2.1 写文本文件



- 运行此程序，会发现在原文件内容后面又追加了重复的内容，这就是将构造方法的第二个参数设为true的效果
- 如果将文件属性改为只读属性，再运行本程序，就会出现IO错误，程序将转入catch块中，给出出错信息

6.2.1 写文本文件



- BufferedWriter类
  - 如果需要写入的内容很多，就应该使用更为高效的缓冲器流类BufferedWriter
  - FileWriter和BufferedWriter类都用于输出字符流，包含的方法几乎完全一样，但BufferedWriter多提供了一个newLine()方法用于换行
    - 不同厂家生产的计算机 (IBM, Apple, VAX, Sun) 对文字的换行方法不同。
    - newLine()方法可以输出在当前计算机上正确的换行符

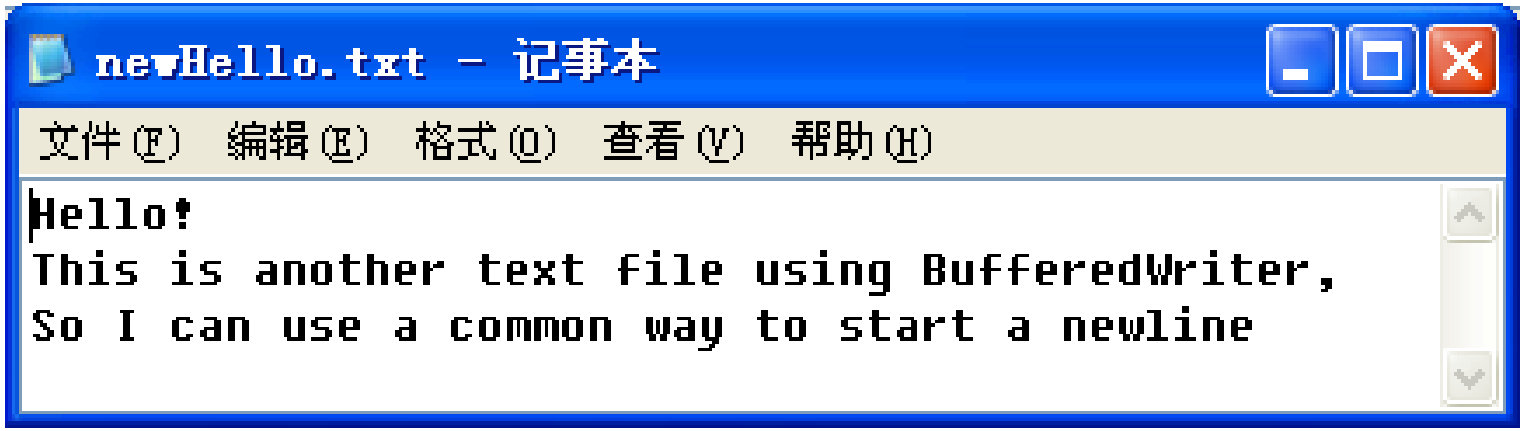
6.2.1 写文本文件

- 使用BufferedWriter完成例WriteTxtFile实现的功能

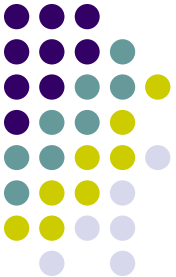
```
import java.io.*;  //BufferedWriter
class BufferWriterEx {
        public static void main ( String[] args ) throws IOException   {
            String fileName = "C:\newHello.txt" ;
            BufferedWriter out = new BufferedWriter(
                                                new  FileWriter( fileName ) );
            out.write( "Hello!"  );
            out.newLine() ;
            out.write( "This is another text file using BufferedWriter,"  );
            out.newLine();
            out.write( "So I can use a common way to start a newline" );
            out.close();
   }
}
```

6.2.1 写文本文件

- 用任何文本编辑器打开newHello.txt都会出现正确的换行效果



第六章 输入/输出流和文件

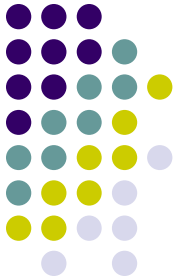


6.1 输入/输出流

6.2 文件读写

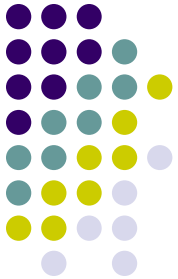
- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.2 读文本文件



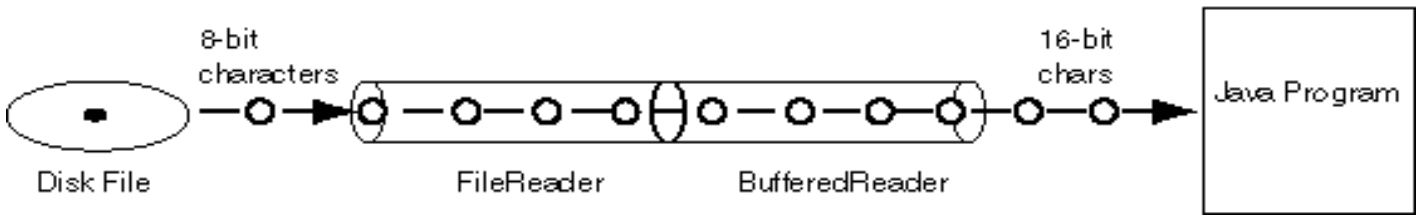
- **本节知识点**
  - **Reader**
  - **FileReader**
  - **BufferedReader和readLine**
  - **文本文件复制**

6.2.2 读文本文件

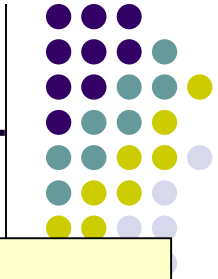


- FileReader类
  - 从文本文件中读取字符
  - 继承自Reader抽象类的子类InputStreamReader
- BufferedReader类
  - 读文本文件的缓冲器类
  - 具有readLine()方法，可以对换行符进行鉴别，一行一行地读取输入流中的内容
  - 继承自Reader
- 文件输入方法：

```
BufferedReader in
    = new BufferedReader(new FileReader( fileName) );
```



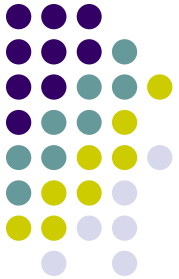
6.2.2 读文本文件



- 从Hello.txt中读取文本并显示在屏幕上

```
import java.io.*;
class ReadTxtFile{
   public static void main ( String[] args ) {
      String fileName = "C:\Hello.txt";
      try { BufferedReader in = new BufferedReader(
                                                new FileReader( fileName  ) );
         line = in.readLine();   //读取一行内容
         while ( line != null ) {
               System.out.println( line );
               line = in.readLine();
          }
        in.close();
      }
      catch ( IOException iox ) {
         System.out.println("Problem reading " + fileName );
      }
   }
}
```

6.2.2 读文本文件



- 说明
- 运行该程序，屏幕上将逐行显示出Hello.txt文件中的内容
- FileReader对象：创建后将打开文件，如果文件不存在，会抛出一个IOException
- BufferedReader类的readLine()方法：从一个面向字符的输入流中读取一行文本。如果其中不再有数据，返回null
- Reader类的read()方法：也可用来判别文件结束。该方法返回的一个表示某个字符的int型整数，如果读到文件末尾，返回 -1。据此，可修改本例中的读文件部分：

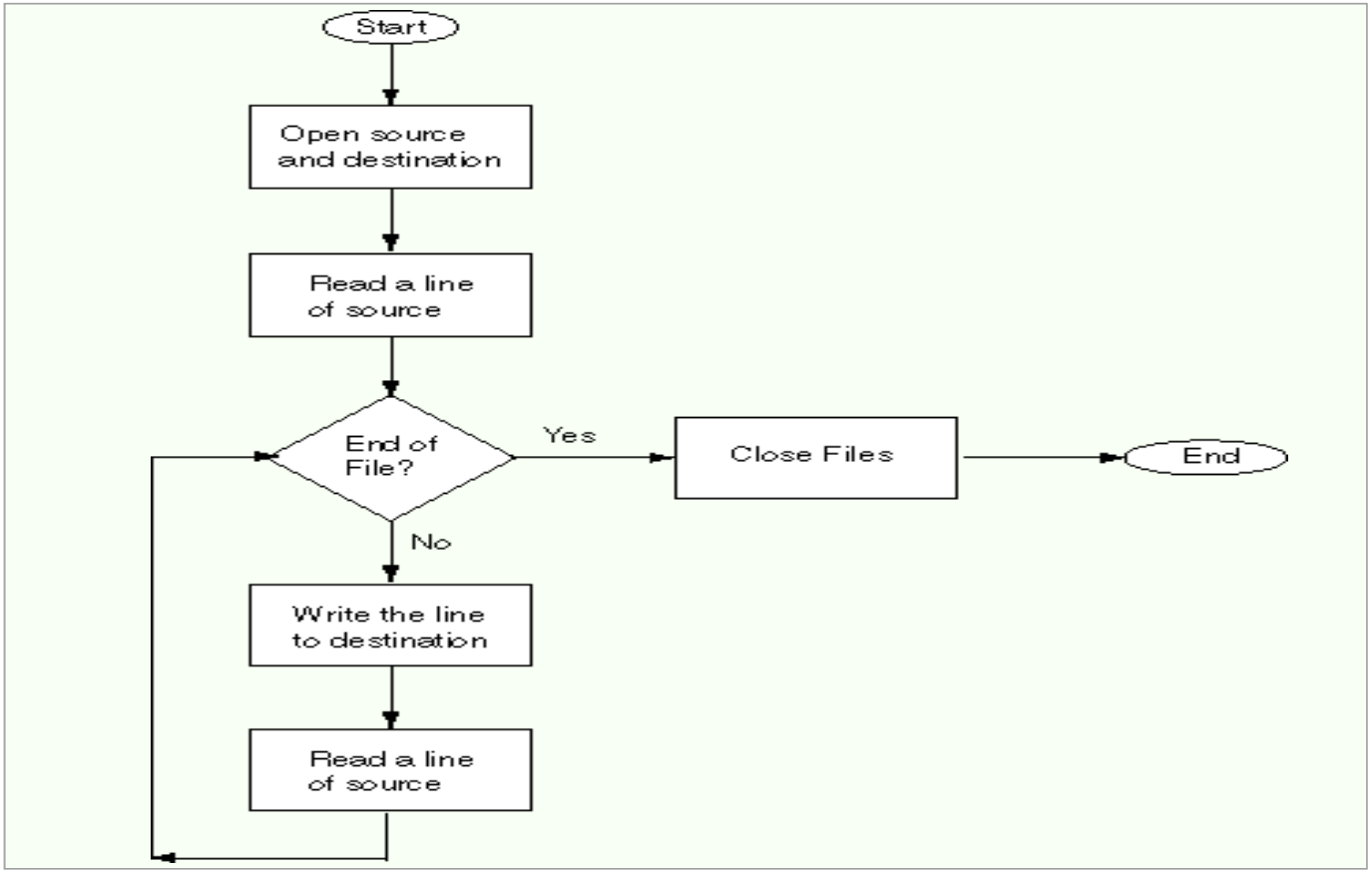
```
int c;
    while((c=in.read())!= -1)
System.out.print((char)c);
```

- close()方法：为了操作系统可以更为有效地利用有限的资源，应该在读取完毕后，调用该方法

6.2.2 读文本文件

- 指定源文件和目标文件名，将源文件的内容拷贝至目标文件。调用方式为：

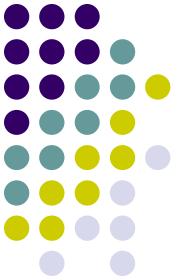
  java copy sourceFile destinationFile



6.2.2 读文本文件

- 例子
  - java CopyFileEx sourcefile destinationfile
- 共包括两个类
  - **CopyMaker**
    - **private boolean openFiles()**
    - **private boolean copyFiles()**
    - **private boolean closeFiles()**
    - **public boolean copy(String src, String dst)**
  - **CopyFileEx**
    - **main()**

6.2.2 读文本文件



- 例子

```
import java.io.*;
class CopyMaker {
    String sourceName, destName;
    BufferedReader source;
    BufferedWriter dest;
    String line;
```

```
private boolean openFiles() {
        try {
                source = new BufferedReader(new FileReader( sourceName ));
        }
        catch ( IOException iox ) {
                System.out.println("Problem opening " + sourceName );
                return false;
    }
    try {
                dest = new BufferedWriter(new FileWriter( destName ));
    }
    catch ( IOException iox )
    {
                System.out.println("Problem opening " + destName );
                return false;
    }
    return true;
 }
```

```
private boolean copyFiles() {
        try {
                line = source.readLine();
                while ( line != null ) {
                   dest.write(line);
                   dest.newLine();
                   line = source.readLine();
                 }
     }
    catch ( IOException iox ) {
                System.out.println("Problem reading or writing" );
                 return false;
     }
    return true;
 }
```

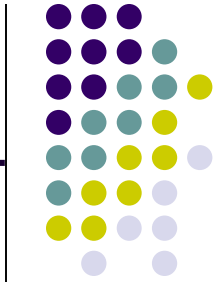
```
private boolean closeFiles() {
    boolean retVal=true;
    try {  source.close();   }
    catch ( IOException iox ) {
              System.out.println("Problem closing " + sourceName );
              retVal = false;
     }
     try {  dest.close();   }
     catch ( IOException iox ) {
               System.out.println("Problem closing " + destName );
               retVal = false;
      }
      return retVal;
 }
  public boolean copy(String src, String dst ) {
       sourceName = src ;
       destName   = dst ;
       return openFiles() && copyFiles() && closeFiles();
    }
}
```

```
public class CopyFileEx    //一个文件中只能有一个公有类
{
      public static void main ( String[] args ) {
            if ( args.length == 2 )
                new CopyMaker().copy(args[0], args[1]);
            else   System.out.println("Please Enter File names");
       }
}
```

- 此文件CopyFileEx.java编译后生成CopyFileEx.class和CopyMaker.class两个字节码文件
- 运行结果
  - 在命令行方式下执行如下命令

    java CopyFileEx c:\Hello.txt c:\CopyHello.txt
  - 则在C盘根目录下会出现CopyHello.txt文件，内容与Hello.txt完全相同

第六章 输入/输出流和文件



6.1 输入/输出流

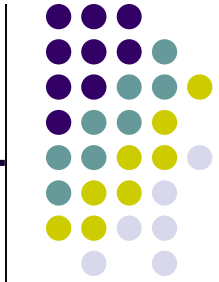
6.2 文件读写

- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.3 写二进制文件

- 本节知识点
  - 二进制文件
  - OutputStream
  - FileOutputStream
  - BufferedOutputStream
  - DataOutputStream
    - writeInt()
    - writeDouble()
    - writeBytes()

6.2.3 写二进制文件

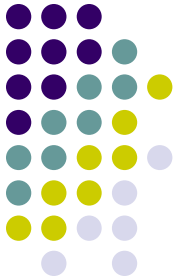


- 二进制文件
  - 原则上讲，所有文件都是由8位的字节组成的
  - 如果文件字节中的内容应被解释为字符，则文件被称为文本文件；
  - 如果被解释为其它含义，则文件被称为二进制文件
  - 例如文字处理程序，例如字处理软件Word产生的doc文件中，数据要被解释为字体、格式、图形和其他非字符信息。因此，这样的文件是二进制文件，不能用Reader流正确读取。
- 为什么需要二进制文件
  - 输入输出更快
  - 比文本文件小很多
  - 有些数据不容易被表示为字符

6.2.3 写二进制文件

- 抽象类OutputStream
  - 派生类FileOutputStream
    - 用于一般目的输出（非字符输出）
    - 用于成组字节输出
  - 派生类DataOutputStream
    - 具有写各种基本数据类型的方法
    - 将数据写到另一个输出流
    - 它在所有的计算机平台上使用同样的数据格式
    - 其中size方法，可作为计数器，统计写入的字节数

6.2.3 写二进制文件



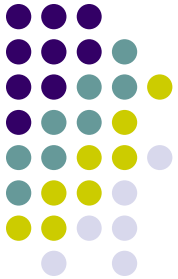
- 将三个int型数字255/0/－1写入数据文件data1.dat

```
import java.io.*;
class WriteByteFileEx{
  public static void main ( String[] args ) {
     String fileName = "c:/data1.dat" ;
     int value0  = 255, value1  = 0, value2 = -1;
     try {
       DataOutputStream out = new DataOutputStream(
                                 new FileOutputStream( fileName  ) );
       out.writeInt( value0 );
       out.writeInt( value1 );
       out.writeInt( value2 );
       out.close();
     }
     catch ( IOException iox ){
            System.out.println("Problem writing " + fileName );   }
     }
 }
```

6.2.3 写二进制文件

- 运行结果
  - 运行程序后，在C盘生成数据文件data1.dat
  - 用写字板打开没有任何显示
  - 用UltraEdit打开查看其二进制信息，
    内容为00 00 00 FF 00 00 00 00 FF FF FF FF，
    每个int数字都是32个bit的
- 说明
  - FileOutputStream类的构造方法负责打开文件“data1.dat”用于写数据
  - FileOutputStream类的对象与DataOutputStream对象连接，写基本类型的数据

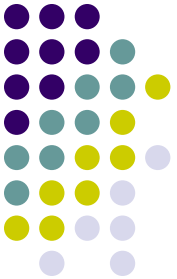
6.2.3 写二进制文件



- BufferedOutputStream类
  - 写二进制文件的缓冲流类
  - 类似于文本文件中的BufferedWriter
  - 对于大量数据的写入，可提高效率
  - 用法示例：

```
DataOutputStream out = new DataOutputStream(
                    new BufferedOutputStream(
                        new FileOutputStream( fileName ) ) );
```

6.2.3 写二进制文件



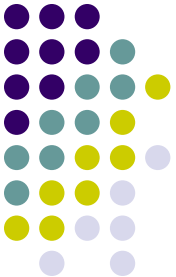
- 向文件中写入各种数据类型的数，并统计写入的字节数

```
import java.io.*;
class WriteByteFileEx2{
  public static void main ( String[] args ) throws IOException {
            String fileName = "mixedTypes.dat" ;
            DataOutputStream dataOut = new DataOutputStream(
                        new BufferedOutputStream(
                          new FileOutputStream( fileName  ) ) );
            dataOut.writeInt( 0 );
            System.out.println( dataOut.size()  + " bytes have been written.");
            dataOut.writeDouble( 31.2 );
            System.out.println( dataOut.size()  + " bytes have been written.");
            dataOut.writeBytes(“Java");
            System.out.println( dataOut.size()  + " bytes have been written.");
            dataOut.close();
  }
}
```

6.2.3 写二进制文件

- 运行结果
  - 4 bytes have been written
  - 12 bytes have been written
  - 16 bytes have been written
- 说明
  - 这个程序可作为字节计数器

第六章 输入/输出流和文件

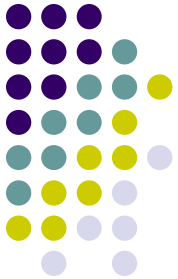


6.1 输入/输出流

6.2 文件读写

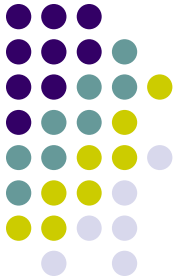
- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.4 读二进制文件



- 本节知识点
  - FileInputStream
  - DataInputStream
  - BufferedInputSteam
  - 读写整数
  - 读写单字节

6.2.4 读二进制文件



- 过滤流
  - 读或写的同时对数据进行处理
  - 通过另外一个流来构造一个过滤流
  - 大部分java.io 包所提供过滤流都是FilterInputStream和FilterOutputStream的子类
    - DataInputStream 和 DataOutputStream
    - BufferedInputStream 和 BufferedOutputStream
    - LineNumberInputStream
    - PushbackInputStream
    - PrintStream

6.2.4 读二进制文件

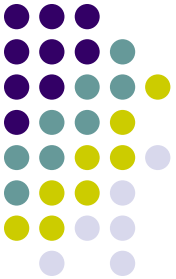
- 读取例WriteByteFileEx类创建的数据文件中的3个int型数字，显示相加结果

```
import java.io.*;
class ReadByteFileEx {
      public static void main ( String[] args ) {
              String fileName = "c:/data1.dat";
              int sum = 0;
       try {  DataInputStream instr = new DataInputStream(
                  new BufferedInputStream(new FileInputStream( fileName )));
               sum += instr.readInt();
               sum += instr.readInt();
               sum += instr.readInt();
               System.out.println( "The sum is: " + sum );
               instr.close();         }
         catch ( IOException iox ) {
                  System.out.println("Problem reading " + fileName ); }
        }
}
```

6.2.4 读二进制文件

- 该程序显示结果是254
- 分析
  - readInt方法可以从输入流中读入4个字节并将其当作int型数据
  - 由于知道文件中存储的是3个int型数据，所以使用了3个读入语句
  - 如果不知道数据的个数该怎么办呢？
  - 因为DataInputStream的读入操作如遇到文件结尾就会抛出EOFException异常，所以我们可以将读操作放入try块中。

6.2.4 读二进制文件



- 修改ReadByteFileEx类
  - 将读操作放入try块中，使遇到文件结尾就会抛出EOFException异常，进入到相应的catch块中

```
try
{
        while ( true )
              sum += instr.readInt();
}
catch ( EOFException  eof )
{
        System.out.println( "The sum is: " + sum );
        instr.close();
}
```

6.2.4 读二进制文件

- 继续修改ReadByteFileEx类
  - 如果没有读到结尾，在读取过程中发生的异常属于IOException，这样就需要我们再加一个catch块处理这种异常
  - 一个try块后面可以跟不止一个catch块，用于处理各种可能发生的异常
  - 我们可以在上段代码后再加上用于捕捉IOException的代码段如下

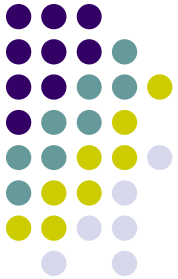
```
catch ( IOException  eof )
{
   System.out.println( "Problem reading input" );
   instr.close();
 }
```

6.2.4 读二进制文件

- 如果catch块中的close方法也发生异常，现在就没法捕获了。解决方法可以有
  - 在main方法中抛出异常
    - 比较简单
    - 缺点是没有catch块，因而无法对异常进行进一步处理，例如给出提示信息
  - 使用嵌套的try块

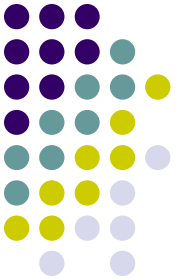
```
import java.io.*;
class ReadByteFileEx 2{
  public static void main ( String[] args ) {
        String fileName = "c:\\data1.dat" ;   long sum = 0;
        try {
                DataInputStream instr = new DataInputStream(
                 new BufferedInputStream(new FileInputStream(fileName)));
              try {
                     while ( true )
                     sum += instr.readInt();
               }
               catch ( EOFException  eof ) {
                      System.out.println( "The sum is: " + sum );
                      instr.close();
                }
          }
          catch ( IOException iox ) {
                System.out.println("IO Problems with " + fileName ); }
           }
       }
}
```

6.2.4 读二进制文件



- 读写字节
- DataOutputStream的writeByte方法
  - public final void writeByte(int b) throws IOException
  - 将int的最不重要字节写入输出流
- DataInputStream的readUnsignedByte方法
  - public final int readUnsignedByte() throws IOException
  - 从输入流中读取1字节存入int的最不重要字节

6.2.4 读二进制文件

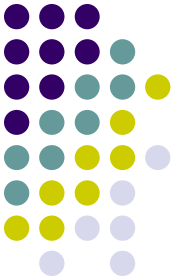


- 从命令行输入源文件名和目标文件名，将源文件复制为目标文件。

```
import java.io.*;
class CopyBytes {
  public static void main ( String[] args ) {
    DataInputStream  instr;
    DataOutputStream outstr;
    if ( args.length != 2 ) {
       System.out.println("Please enter file names");
       return;
    }
    try {
        instr = new DataInputStream(new
                BufferedInputStream(new FileInputStream( args[0] )));
        outstr = new DataOutputStream(new
                    BufferedOutputStream(new FileOutputStream( args[1] )));
```

```
        try {
                int data;
                while ( true ) {
                data = instr.readUnsignedByte() ;
                outstr.writeByte( data ) ;
               }
      }
      catch ( EOFException  eof ) {
            outstr.close();
            instr.close();
            return;
      }
   }
   catch ( FileNotFoundException nfx )
            {   System.out.println("Problem opening files" );    }
   catch ( IOException iox )
            {   System.out.println("IO Problems" );    }
   }
}
```

第六章 输入/输出流和文件

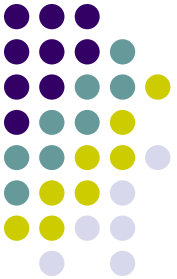


6.1 输入/输出流

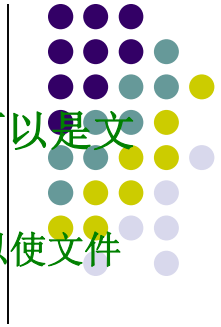
6.2 文件读写

- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.5 File类

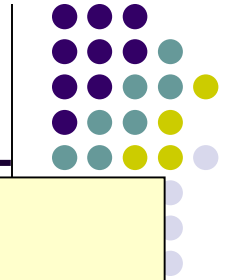


- 表示磁盘文件信息
- 定义了一些与平台无关的方法来操纵文件
  - 创建、删除文件
  - 重命名文件
  - 判断文件的读写权限及是否存在
  - 设置和查询文件的最近修改时间等
- 构造文件流可以使用File类的对象作为参数



java.io.File	
+File(pathname:String)	根据pathname创建File对象。pathname可以是文件或目录
+File(parent:String, child:String)	在parent这个目录下，创建File对象，child可以使文件或目录
+File(parent:File, child:String)	同上，只是parent是一个File对象
+exists():boolean	文件或目录是否存在
+canRead():boolean	是否可读
+canWrite():boolean	是否可写
+isDirectory():boolean	是否为目录
+isFile():boolean	是否为文件
+isAbsolute():boolean	本File对象是否采用绝对路径来创建的
+isHidden():boolean	是否为隐藏文件
+getAbsolutePath():String	得到绝对路径
+getCanonicalPath():String	得到规范的绝对路径
+getName():String	得到文件/目录名
+getPath():String	得到路径
+getParent():String	得到父目录的路径名
+lastModified():long	最后修改时间
+delete():boolean	删除
+renameTo(dest:File):boolean	重命名

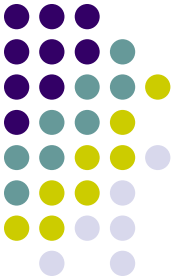
6.2.5 File类



```
public class FileTest {
    public static viod main(String[] args)
    {
        java.io.File file = new java.io.File("image\up.gif");
        System.out.println("Does it exist? " + file.exists());
        System.out.println("Can it be read? " + file.canRead());
        System.out.println("Can it be written? " + file.canWrite());
        System.out.println("Is it a directory? " + file.isDirectory());
        System.out.println("Is it a file? " + file.isFile());
        System.out.println("Is it absolute? " + file.isAbsolute());
        System.out.println("Is it hidden? " + file.isHidden());
        System.out.println("Absolute path is " + file.getAbsolutePath());
        System.out.println("Last modified on "
                    + new java.util.Date(file.lastModified()));
    }
}
```

```
Does it exist? true
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
Absolute path is E:\FreePP\Android\project\java-test\image\up.gif
Last modified on Thu Aug 06 16:41:51 CST 2012
```

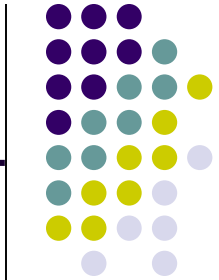
6.2.5 File类



- 在C盘创建文件Hello.txt，如果存在，则删除旧文件，不存在则直接创建新的

```
import java.io.*;
public class FileEx {
    public static void main(String[] args) {
        File f=new File("c:"+File.separator+"Hello.txt");
        if (f.exists())
            f.delete();
        else
            try{
                f.createNewFile();
            }
            catch(Exception e){
                System.out.println(e.getMessage());
            }
    }
}
```

6.2.5 File类



- 运行结果
  - 因为在前例中已经创建了c:\Hello.txt，所以第一次运行将删除这个文件
  - 第二次运行则又创建了一个此名的空文件
- 分析
  - 在试图打开文件之前，可以使用File类的isFile方法来确定File对象是否代表一个文件而非目录）
  - 还可通过exists方法判断同名文件或路径是否存在，进而采取正确的方法，以免造成误操作

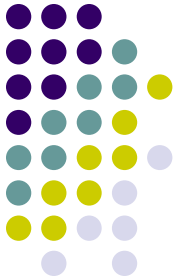
6.2.5 File类--改进的文件复制程序

```
import java.io.*;
class NewCopyBytes{
   public static void main ( String[] args ) {
       DataInputStream  instr;
       DataOutputStream outstr;
       if ( args.length != 2 ) {
          System.out.println("Please Enter file names!");
          return;
       }
      File inFile  = new File( args[0] );
      File outFile = new File( args[1] );
      if ( outFile.exists() ) {
        System.out.println( args[1] + " already exists");
        return;
      }
      if ( !inFile.exists() )  {
        System.out.println( args[0] + " does not exist");
        return;
      }
```

```
  try{
      instr = new DataInputStream(new BufferedInputStream(
                                  new FileInputStream( inFile )));
      outstr = new DataOutputStream(new BufferedOutputStream(
                                  new FileOutputStream( outFile  )));
      try {
            int data;
            while ( true ) {
               data = instr.readUnsignedByte() ;
               outstr.writeByte( data ) ;

             }
       }
      catch ( EOFException  eof )
         {    outstr.close();   instr.close();    return;     }
   }
   catch ( FileNotFoundException nfx )
   {  System.out.println("Problem opening files" );    }
   catch ( IOException iox )
   {  System.out.println("IO Problems" );    }
  }
}
```

第六章 输入/输出流和文件

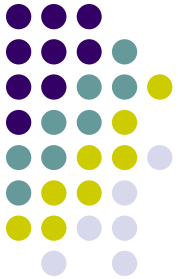


6.1 输入/输出流

6.2 文件读写

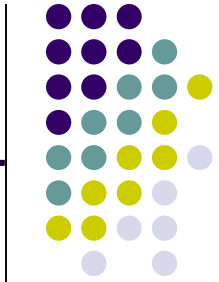
- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.6 对象序列化（串行化）



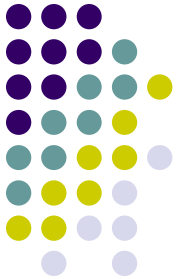
- 对象序列化
  - 概念：为了保存在内存中的各种对象的状态，并且可以把保存的对象状态再读出来。Java提供一种保存对象状态的机制——序列化（串行化）。
  - 目的：a）内存中的对象保存到一个文件中或者数据库中；b）用套接字在网络上传送对象；c）通过RMI传输对象；
  - 任务：写出对象实例变量的数值。如果变量是另一个对象的引用，则引用的对象也要序列化。
  - 方法：只有实现了Serializable接口的类的对象才可以被序列化；该对象必须与一定的对象输入和输出流联系进行状态保存和状态恢复。

6.2.6对象序列化



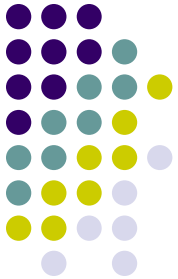
- 保存对象的信息，在需要的时候，再读取这个对象
- 内存中的对象在程序结束时就会被垃圾回收机制清除
- 用于对象信息存储和读取的输入输出流类：
  - ObjectInputStream
  - ObjectOutputStream

6.2.6对象序列化



- ObjectInputStream和ObjectOutputStream
  - 实现对象的读写
    - 通过ObjectOutputStream把对象写入磁盘文件
    - 通过ObjectInputStream把对象读入程序
  - 不保存对象的transient和static类型的变量
  - 对象要想实现序列化，其所属的类必须实现Serializable接口

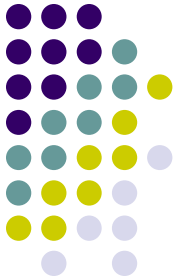
6.2.6 对象序列化



- 写入ObjectOutputStream
- 必须通过另一个流构造ObjectOutputStream:

```
FileOutputStream out = new FileOutputStream("theTime");
ObjectOutputStream s =  new ObjectOutputStream(out);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
s.close();
```

6.2.6 对象序列化



- 用ObjectInputStream读入
- 必须通过另一个流构造ObjectInputStream：

```
FileInputStream in = new FileInputStream("theTime");

ObjectInputStream s = new ObjectInputStream(in);

String today = (String)s.readObject();

Date date = (Date)s.readObject();
```

6.2.6 对象序列化

- Seriealizable接口
- 空接口，使类的对象可实现序列化
- Serializable 接口的定义

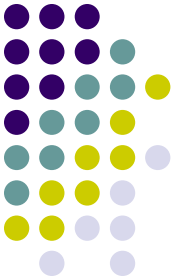
```
package java.io;
public interface Serializable {
    // there's nothing in here!
};
```

- 实现Serializable接口的语句

```
public class MyClass implements Serializable {
    ...
}
```

- 使用关键字transient可以阻止对象的某些成员被自动写入文件

6.2.6 对象序列化



- 对象序列化
  - 注意事项：序列化只能保存对象的实例成员变量的值；状态瞬时的对象不能序列化，保密的字段应加transient关键字。
  - 定制序列化：默认的序列化机制，按照名称的升序写入其数值，如果想控制这些数值的写入顺序和写入类型，必须自己定义读写数据流的方式。

6.2.6 对象序列化

- 创建一个书籍对象，并把它输出到一个文件book.dat中，然后再把该对象读出来，在屏幕上显示对象信息

```
class Book implements Serializable {
        int id;
        String name;
        String author;
        float price;
        public Book(int id,String name,String author,float price) {
                this.id=id;
                this.name=name;
                this.author=author;
                this.price=price;
        }
}
```

```
import java.io.*;
public class SerialEx {
      public static void main(String args[]) throws
                                                IOException,ClassNotFoundException
      {
          Book book=new Book(100032,"Java Programming Skills",
                                        "Wang Sir",30);
          ObjectOutputStream oos = new ObjectOutputStream(
                                                  new FileOutputStream("c:\book.dat"));
          oos.writeObject(book);
          oos.close();
          book=null;
          ObjectInputStream ois = new ObjectInputStream(
                                                    new FileInputStream("c:\book.dat"));
          book=(Book)ois.readObject();
          ois.close();
          System.out.println("ID is:"+book.id);
          System.out.println("name is:"+book.name);
          System.out.println("author is:"+book.author);
          System.out.println("price is:"+book.price);
       }
}
```

6.2.6 对象序列化

- 运行结果

  将生成book.dat文件，并在屏幕显示：

  ID is:100032

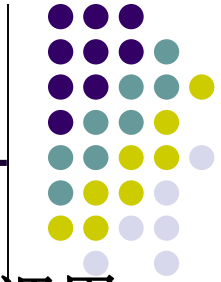
  name is:Java Programming Skills

  author is:Wang Sir

  price is:30.0

- 说明
  - **如果希望增加Book类的功能，使其还能够具有借书方法borrowBook，并保存借书人的借书号borrowerID，可对Book类添加如下内容：**

```
transient int borrowerID;
public void borrowBook(int ID)
{
    this.borrowerID=ID;
}
```



- 在main方法中创建了Book类的一个对象后，紧接着调用borrowBook方法

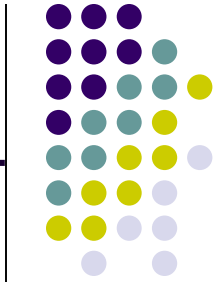
```
book.borrowBook(2012);
```

- 从读入的对象中输出borrowerID

```
System.out.println("Borrower ID is:"+book.borrowerID);
```

- 运行结果
  - 显示borrrowID为0，因为声明为transient，所以不保存
  - 如果去掉transient关键子，则可以正确读出2012。这对于保护比较重要的信息（例如密码等）是很有必要的

第六章 输入/输出流和文件

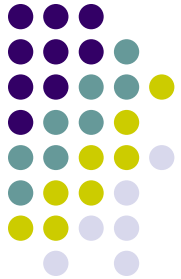


6.1 输入/输出流

6.2 文件读写

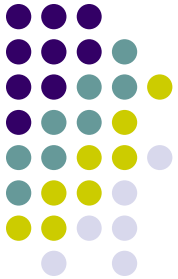
- 写文本文件
- 读文本文件
- 写二进制文件
- 读二进制文件
- File类
- 对象序列化
- 随机文件读写

6.2.7 随机文件读写



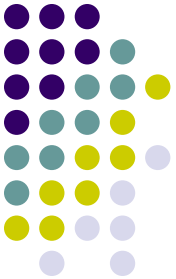
- 随机文件读写
  - 对于很多场合，例如银行系统、实时销售系统，要求能够迅速、直接地访问文件中的特定信息，而无需查找其他的记录。这种类型的即时访问可能要用到随机存取文件和数据库。
  - 随机文件的应用程序必须指定文件的格式。最简单的是要求文件中的所有记录均保持相同的固定长度。利用固定长度的记录，程序可以容易地计算出任何一条记录相对于文件头的确切位置
  - Java.io包提供了RandomAccessFile类用于随机文件的创建和访问

6.2.7 随机文件读写



- **RandomAccessFile类**
  - **可跳转到文件的任意位置读/写数据**
  - **可在随机文件中插入数据，而不破坏该文件的其他数据**
  - **实现了DataInput 和 DataOutput 接口，可使用普通的读写方法**
  - **有个位置指示器，指向当前读写处的位置。刚打开文件时，文件指示器指向文件的开头处。对文件指针显式操作的方法有：**
    - **int skipBytes(int n)：**把文件指针向前移动指定的n个字节
    - **void seek(long)：**移动文件指针到指定的位置。
    - **long getFilePointer()：**得到当前的文件指针。
  - **在等长记录格式文件的随机读取时有很大的优势，但仅限于操作文件，不能访问其它IO设备，如网络、内存映像等**

6.2.7 随机文件读写



- **RandomAccessFile类**
  - **可用来实现读和写，构造方法包括**

    public RandomAccessFile(File file, String mode)
    throws FileNotFoundException

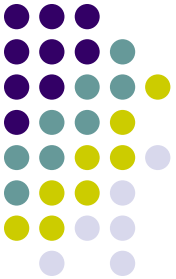
    public RandomAccessFile(String name, String mode)
    throws FileNotFoundException
  - **建立一个RandomAccessFile时，要指出你要执行的操作：仅从文件读，还是同时读写**

    new RandomAccessFile("farrago.txt", "r");

    new RandomAccessFile("farrago.txt", "rw");

```
import java.io.*;
public class ReadRandomFile {      //随机读取文件内容
   public static void readFileByRandomAccess(String fileName) {
        RandomAccessFile randomFile = null;
    try {
        System.out.println("随机读取一段文件内容： ");
        // 打开一个随机访问文件流，按只读方式
        randomFile = new RandomAccessFile(fileName, "r");
        // 文件长度，字节数
        long fileLength = randomFile.length();
        // 读文件的起始位置
        int beginIndex = (fileLength > 4) ? 4 : 0;
        // 将读文件的开始位置移到beginIndex位置。
        randomFile.seek(beginIndex);
        byte[] bytes = new byte[10];
        int byteread = 0;
        // 一次读10个字节，如果文件内容不足10个字节，则读剩下的字节。
        // 将一次读取的字节数赋给byteread
        while ((byteread = randomFile.read(bytes)) != -1) {
            System.out.write(bytes, 0, byteread);
        }
    }
```

```
    catch (IOException e) {
        e.printStackTrace();
      }
      finally {
        if (randomFile != null) {
          try {  randomFile.close();  }
          catch (IOException e1) { System.out.println("IO Problems");  }
        }
      }
   }
}
```



- 本章内容
  - **I/O流的概念以及分类**
  - **读写文本文件、二进制文件的方法**
  - **处理流的概念及用法**
  - **File类**
  - **对象序列化的常用流类及接口**
  - **随机读写文件的流类**
- 本章要求
  - **理解I/O流的概念，掌握其分类**
  - **掌握文本文件读写、二进制文件读写、处理流类的概念和用法、对象序列化**
  - **掌握File类、随机读写流类**
  - **遇到I/O方面的问题，能够自行查阅API文档解决**