第8讲 Java异常处理及输入输出流简介

- 8.1 异常处理概述
- 8.2 Java异常的处理方法
- 8.3 Java数据流概述和java.io包
- 8.4 File类
- 8.5 文件输入与输出
- 8.6 标准输入和输出

本章要点

- 了解异常和异常分类
- 理解Java异常处理机制和异常类。
- 掌握try/catch/ finally语句处理异常的方式以及如何声明异常。
- 了解自定义异常。
- 理解Java语言中数据流的概念
- 掌握各种常用的输入输出流类
- 掌握各种文件操作类，
- 了解流类的继承关系

8.1 异常处理概述

Java异常是描述在代码段中发生的运行出错情况的对象。程序中的错误可能来自于编译错误和运行错误。编译错误是由于所编写的程序存在语法问题，未能通过由源代码到目标代码的编译过程而产生的错误，它将由语言的编译系统负责检测和报告；运行错误是在程序的运行过程中产生的错误。

8.1.1 什么是异常

在程序执行中，任何中断正常程序流程的异常条件就是错误或异常。

第一种，Java 虚拟机检测到了非正常的执行状态，这些状态可能是由以下几种情况引起的：

① 表达式的计算违反了Java 语言的语义，例如整数被0 除。

② 在载入或链接Java 程序时出错。

③ 超出了某些资源限制，例如使用了太多的内存。

第二种，Java 程序代码中的throw 语句被执行。

第三种，异步异常发生。异步异常的原因可能有：

① Thread 的stop 方法被调用。

② Java 虚拟机内部错误发生。

8.1.2 异常处理机制

Java异常处理是通过5个关键字来管理的。它们是try、catch、throw、throws和finally。

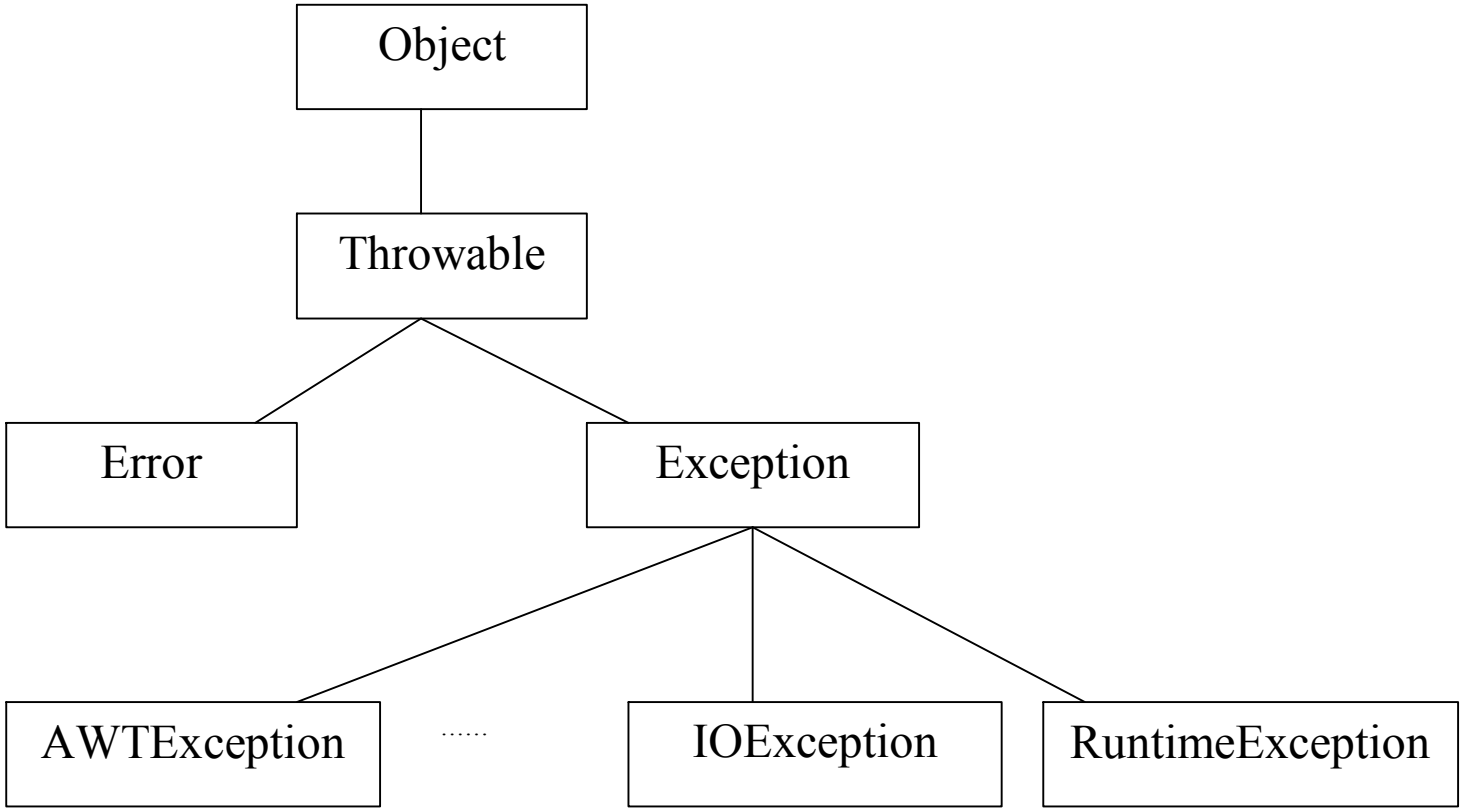
程序里，需要被监测的程序语句序列应包含在一个try代码块中。如果try代码块中有异常发生，那么就要抛出该异常。可以用catch来捕获这个异常，并且在catch块中加以适当地处理。系统产生的异常会由Java运行时系统自动抛出。如果要手动抛出异常，则使用关键字throw。在一些情况下，从一个方法抛出的异常必须用一个throws语句指定为异常。最后，从try代码块退出时，必须执行的代码要放在一个finally代码块中。

8.1.3 异常分类

Java中的异常类具有层次组织，其中Throwable类是Error类（错误类）和Exception类（异常类）的父类，Throwable是Object的直接子类。

异常类（java.lang.Exception）继承于java.lang.Object中的java.lang.Throwable类。异常可分为执行异常（RuntimeException）和检查异常（Checked Exceptions）两种。

异常类的继承结构


Object
Throwable
Error
Exception
AWTException
.....
IOException
RuntimeException

1. 执行异常

即运行时异常，继承于RuntimeException。Java编译器允许程序不做处理。

ArithmeticException：一个不寻常算术运算产生的异常。

ArrayStoreException：存入数组的内容数据类型不一致所产生的异常。

ArrayIndexOutOfBoundsException：数组索引超出范围所产生的异常。

ClassCastExcption：类对象强迫转换造成不当类对象所产生的异常。

NumberFormatException：字符串转换数值所产生的异常。

IndexOutOfBoundsException：索引超出范围所产生的异常。

NegativeException：数组建立负值索引所产生的异常。

NullPointerException：对象引用参考值为null所产生的异常。

2. 检查异常

除了执行异常外，其余的子类是属于检查

异常类也称为非运行时异常，它们都在

java.lang类库内定义。Java编译器要求程序必须捕获或者声明抛弃这种异常。

ClassNotFoundException：找不到类或接口所产生的异常。

CloneNotSupportedException：使用对象的clone方法但无法执行Cloneable所产生的异常。

IllegaAccessException：类定义不明确所产生的异常。

InstantiationException：使用newInstance方法试图建立一个类instance时所产生的异常。

InterruptedException：目前线程等待执行，另一线程中断目前线程所产生的异常。

8.1.4 错误分类

Error类与异常一样，它们都是继承自**java.lang.Throwable**类。**Error** 类对象由**Java** 虚拟机生成并抛出。**Error**类包括**linkageError**（结合错误）与**VitualmachineError**（虚拟机错误）两种子类。

1．linkageError

LinkageError 的子类表示一个类依赖于另一个类，但是，
在前一个类编译之后， 后一个类的改变会与它不兼容。
ClassFormarErro：类格式所产生的错误。
ClassCircularityError：无限循所产生的错误。
ExceptionInInitializerError：初始化所产生的错误。
NoClassDeFormatError：没有类定义所产生的错误。
VeritfyError：类文件某些数据不一致或安全问题所产生的错误。
UnsatisfidLinkError：Java虚拟机无法找到合适的原始语言（native-language）
定义的方法所产生的错误。
IncompatibleClassChangeError：不兼容类所产生的错误。
AbtractMethodError：调用抽象方法所产生的错误。
NoSuchFieldError：存取或改变数据域所产生的错误。
NoSchMethodError ：调用类方法所产生的错误。
IllegalAccessError：不合法存取或改变数据域或调用方法所产生的错误。
InstantionError：使用抽象类或接口所产生的错误。

2．VitualmachineError

当Java虚拟机崩溃了或用尽了它继续操作所需的资源时，抛出该错误。

VitualmachineError包含InternalError，OutOfMemoryError ，StackOverflow- Error，UnknownError。这些类所代表的意义：

InternalError：虚拟机内部所产生的错误。

OutOfMemoryError：虚拟机内存不足所产生的错误。

StackOverflowError：堆栈无法容纳所产生的错误。

UnknownError：虚拟机不知名异常所产生的错误。

8.2 Java异常的处理方法

当发生Exception时，那么，在编写程序时可以采取的措施：

① 通过将try｛｝catch（）｛｝块纳入其代码中，在这里捕获被命名为属于某个超类的异常，并调用方法处理它。

② 让被调用的方法表示它将不处理异常，将该异常抛到它所遇到的调用方法中。这用throws子句实现。

使用throws子句标记调用方法的声明如下所示：

```
public void troublesome() throws Exception
```

其中，关键字throws之后是所有异常的列表，方法可以将它们抛回到它的调用程序中。

8.2.1 try/catch/finally

1．try/catch

try/catch异常处理代码块的基本形式：

```
  try      //监视
{
可能发生异常的代码块;
}
  catch（异常类型  异常对象名）//捕获并处理异常
{
异常处理代码块;
}
```

当抛出一个异常时，异常会由相应的catch语句捕获并处理。与一个try相关的catch语句可以有多个，构成多重catch 语句，异常类型决定了要执行哪个catch语句。

如果没有抛出异常，那么try代码块就会结束，并且会跳过它的所有catch语句，从最后一个catch后面的第一个语句继续执行。

例8.1使用try/catch进行异常处理的情况。

```
public class  ExceptionExam0
{
  public static void main(String args[])
{
    int i,a;
    try { // 监视一代码块
                        i=0;
                        a=42/i;
return;
        }
catch (ArithmeticException e)
{ //捕获一个被零除异常
System.out.println("被零除");
}
  }
}
```

例8.1运行结果如图所示

2. 可嵌入的try块

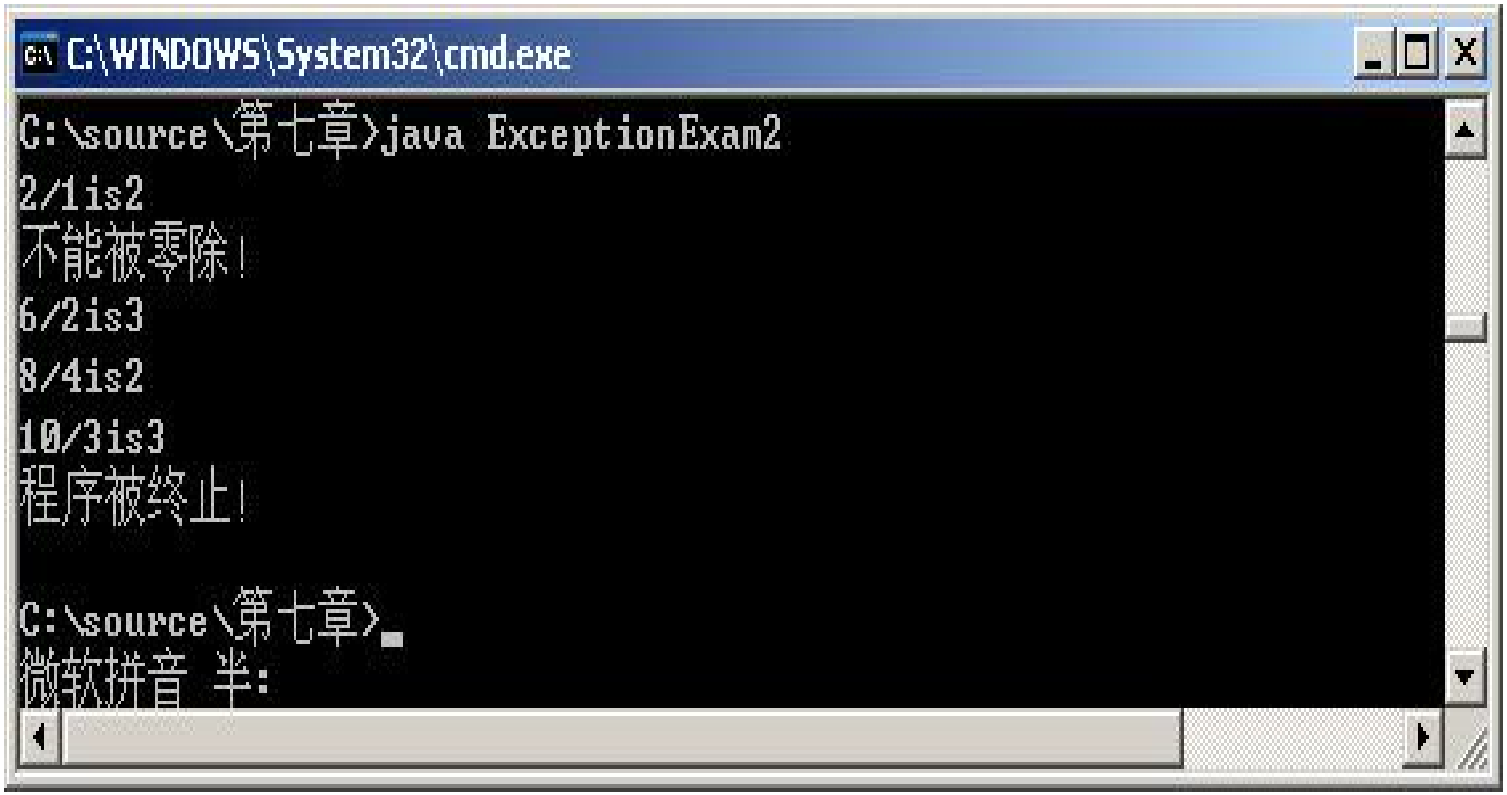
一个try代码块可以嵌入到另一个try代码块当中。由于内部try代码块产生的异常如果没有被与该内部try代码块相关的catch捕获，就会传到外部try代码块。

通常嵌入式try代码块用于以不同方式处理不同类型的错误。某些类型的错误是致命的，无法修改。某些错误则较轻，可以马上处理。许多程序员在使用外部try代码块捕获大部分严重错误的同时，让内部try代码处理不太严重的错误。

例8.2　嵌套try语句的示例。

```
public class ExceptionExam2
{
public static void main(String args[])
{
int data1[ ]={2,4,6,8,10,12};
int data2[ ]={1,0,2,4,3};
try
{
for(int i=0;i<data1.length;i++) {
try{
System.out.println(data1[i]+ "/"+data2[i]+
     "is"+data1[i]/data2[i]);
}
catch(ArithmeticException e)
{
System.out.println("不能被零除！");
}}}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println("程序被终止！");
}}}
```

例8.2运行结果如图所示



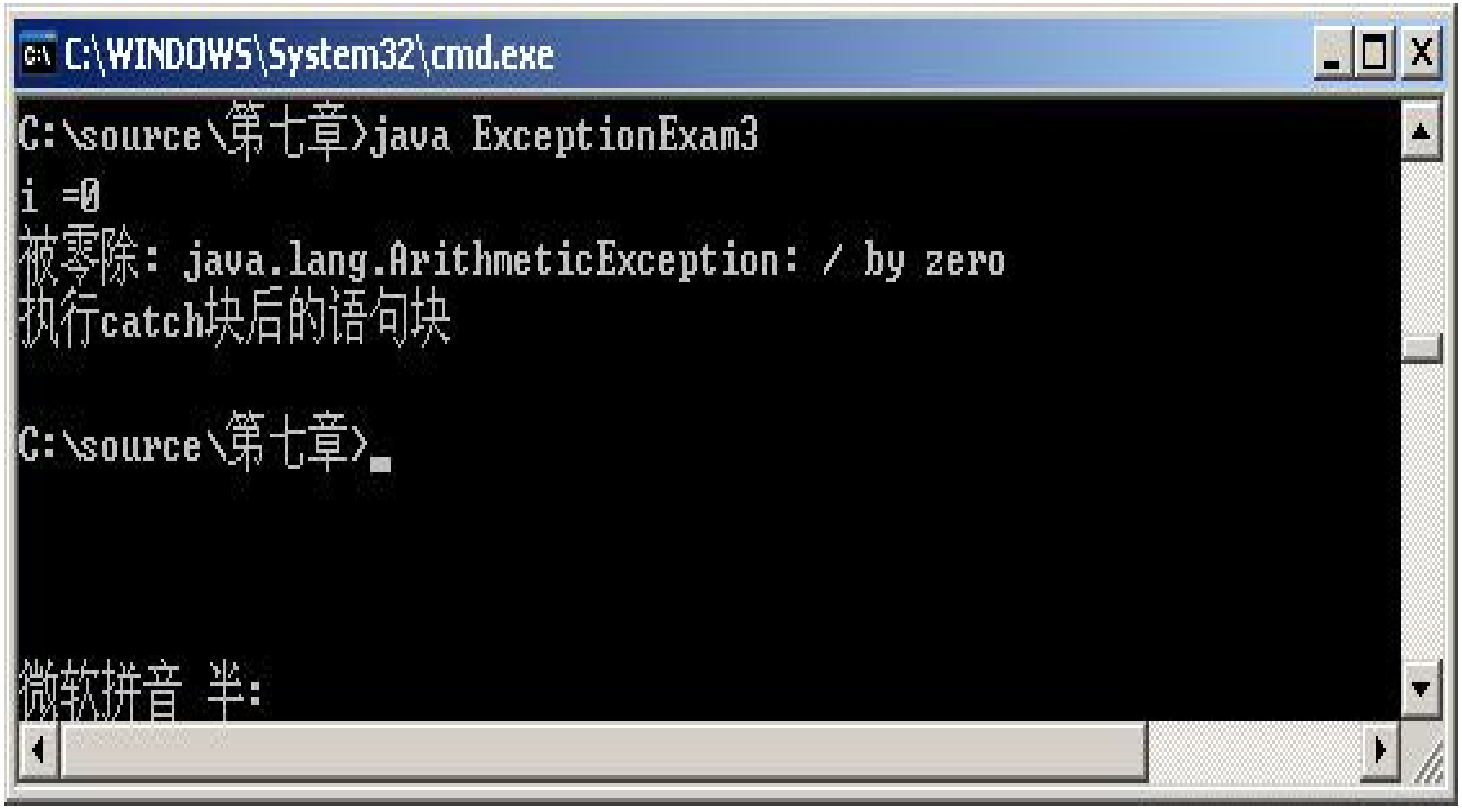
3．使用多重catch 语句

与一个try相关的catch语句可以有多个。每一个catch语句捕获一个不同类型的异常。某些情况，由单个代码段可能引起多个异常。处理这种情况时就需要定义两个或更多的catch子句，每个子句捕获一种类型的异常。当异常被引发时，每一个catch子句被依次检查，第一个匹配异常类型的子句被执行。当一个catch语句执行以后，其他的子句被忽略，程序从try/catch块后的代码开始继续执行。

例8.3　捕获两种不同类型的异常。

```
public class ExceptionExam3
 {
  public static void main(String args[])
 {
try
{
int i = args.length;
System.out.println("i ="+i);
int j=5/i;
int k[]={ 1,2,3 };
k[5]=0;
}
catch(ArithmeticException e)
{
 System.out.println("被零除: " + e);
}
catch(ArrayIndexOutOfBoundsException e)
{
 System.out.println("Array index out of bound exception: " + e);
}
 System.out.println("执行catch块后的语句块");
}}
```

例8.3运行结果如图所示



4．finally关键字的使用

try/catch/finally的基本形式如下所示：

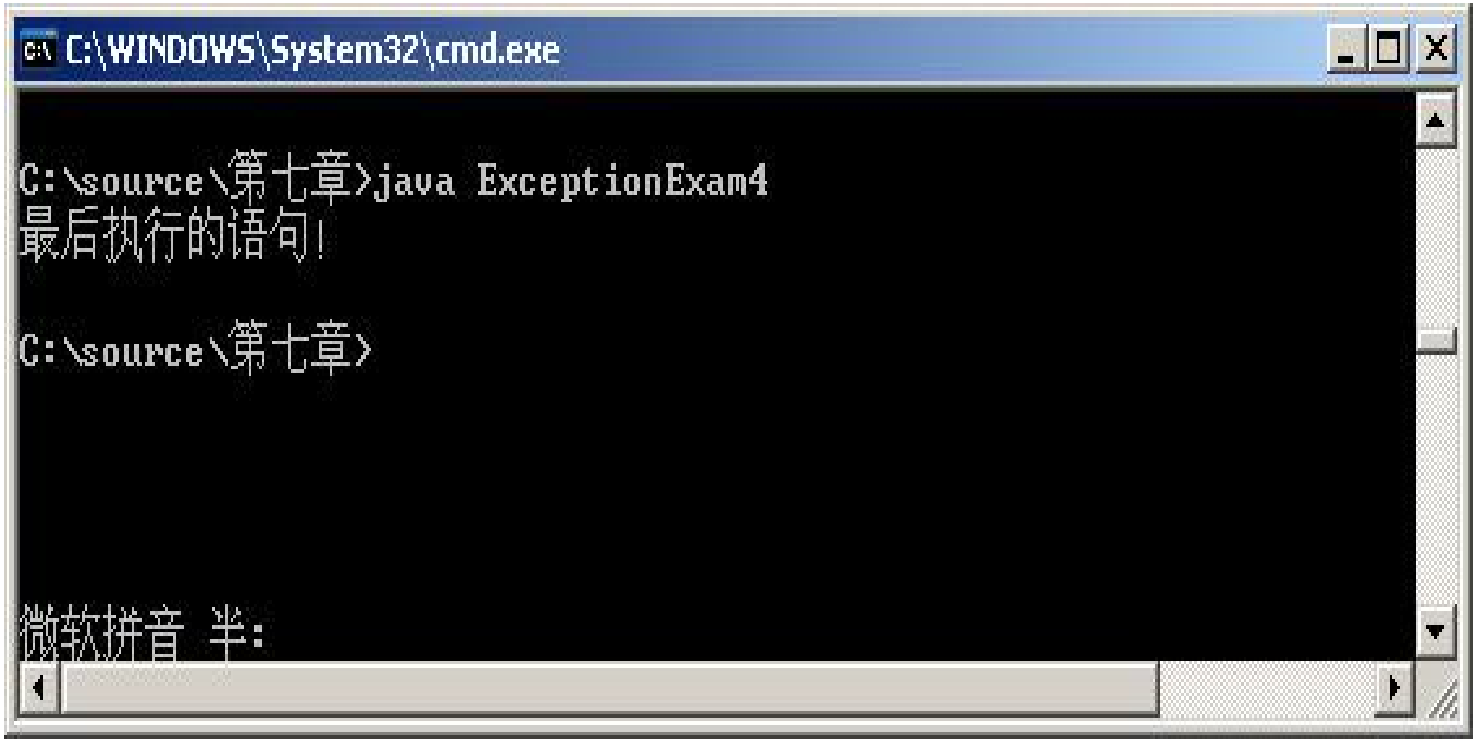
```
try
{
可能发生异常的代码块;
}
catch（异常类型  异常对象名）
{
异常处理代码块;
}
…
finally
{
无论是否抛出异常都要执行的代码;
}
```

无论是出于何种原因，只要执行离开try/catch代码块，就会执行finally代码块。即无论try是否正常结束，都会执行finally定义的最后的代码。

例8.4　使用finally的示例。

```
public class ExceptionExam4
{
    public static void main(String [] args)
     {
          try
            {
                    int [] a=new int[3];
              a[2]=4;
return;
            }
catch(ArithmeticException e)
{
          System.out.println("发生了异常");
}
finally
{
          System.out.println("最后执行的语句！");
}}}
```

例8.4运行结果如图所示



8.2.2 声明异常（throws）

声明抛弃异常是在一个方法声明中的**throws**子句中指明的。

下面是包含**throws**子句的方法的基本形式：

[修饰符] 返回类型 方法名（参数1,参数2,……）throws 异常列表

{……}

例如：

```
public int read () throws IOException
{……}
```

throws子句中同时可以指明多个异常，说明该方法将不对这些异常进行处理，而是声明抛弃它们。例如：

```
public static void main(String args[]) throws
    IOException, IndexOutOfBoundsException
{……}
```

8.2.3 抛出异常（throw）

手动抛弃异常对象是通过throw语句实现的，但可以抛弃的异常必须是Throwable或其子类的实例。其基本形式如下：

throw 异常名;

throw关键字主要是用在try块中，用来说明已经发生的异常情况。throw关键字后面跟随一个从类Throwable中派生的异常对象，用来说明发出的异常类型。throw语句促使程序立即停止运行，并且重复执行最近能够处理指定对象的catch语句。如果异常在程序的其他地方产生，throw语句也可以放在try语句的后面。为了把异常处理控制传递给更高层的处理模块，还可以对截获的异常对象再一次实施throw操作。

例8.5 使用throw关键字手动抛出IOException异常。

```
import java.io.*;
public class ExceptionExam6
{
public static void main(String [] args)
{
try
{
System.out.println("…正在运行程序…");
throw new IOException("用户自行产生异常");
}
catch(IOException e)
{
System.out.println("已捕获了该异常！ ");
}
}
}
```

例8.5运行结果如图所示

8.2.4 自定义Java异常

若要建立自己的异常类型，只要定义Exception的一个子类就可以了，子类不需要实际执行什么——它们在类型系统中的存在允许把它们当成异常使用。

自定义异常的基本形式如下所示：

```
class 自定义异常  extends 父异常类名
 {
类体;
}
```

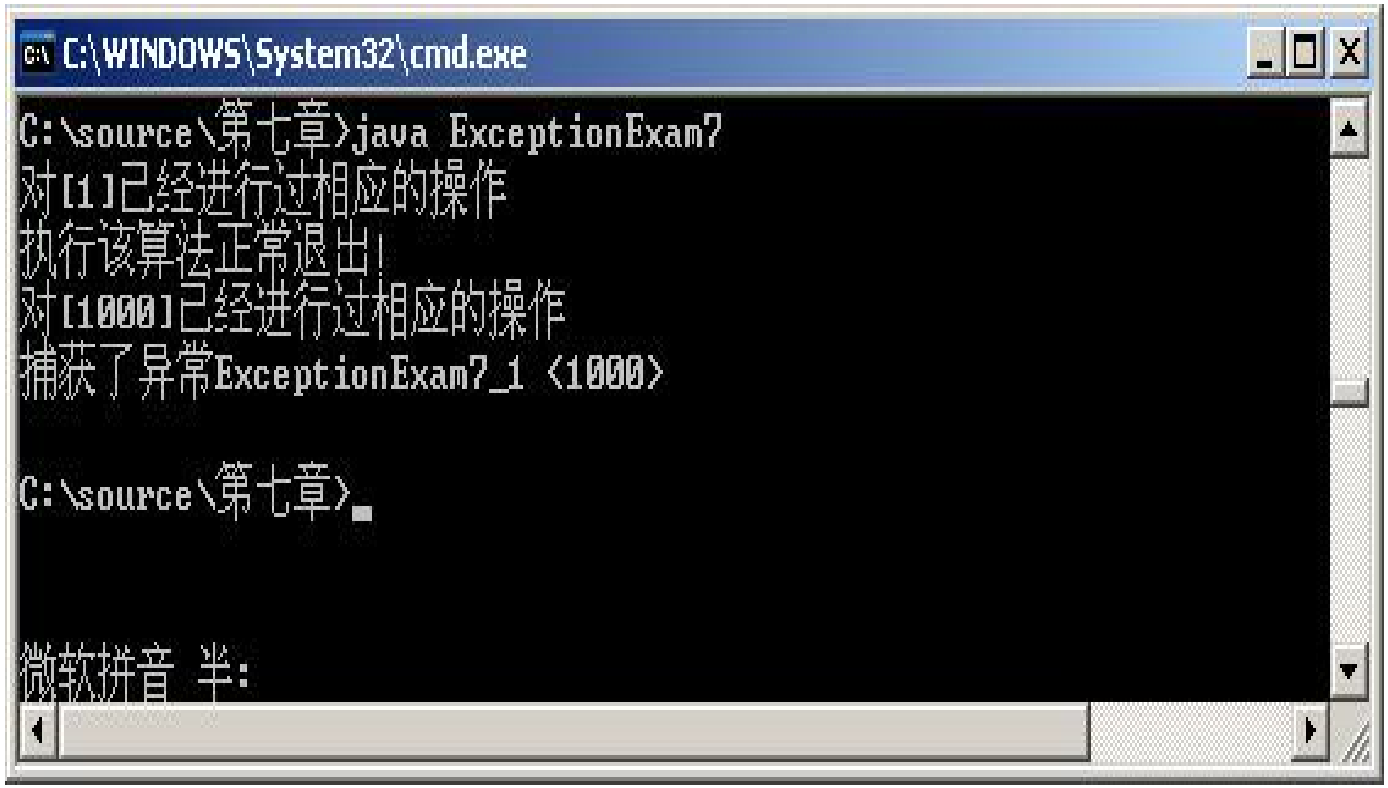
例8.6 自定义异常示例

```
class ExceptionExam7_1 extends Exception        //自定义异常类
    ExceptionExam7_1
{
private int show;
ExceptionExam7_1 (int a)
{show=a;}
public String toString()
{return "ExceptionExam7_1 <"+show+">";}}
public class ExceptionExam7
{static void caculate(int a) throws ExceptionExam7_1
{
System.out.println("对["+ a +"]已经进行过相应的操作");
if(a>100)    throw new ExceptionExam7_1 (a);
```

续

```
System.out.println("执行该算法正常退出！ ");
}
public static void main(String args[])
{
try{
caculate(1);
caculate(1000);
}catch (ExceptionExam7_1 e)
{
System.out.println("捕获了异常" + e);
}}}
```

例8.6运行结果如图所示



8.3 Java数据流概述和java.io包

8.3.1 流的概念

所谓流是指同一台计算机或网络中不同计算机之间有序运动着的数据序列。Java把这些不同来源和目标的数据都统一抽象为数据流。数据流可分为输入流和输出流。

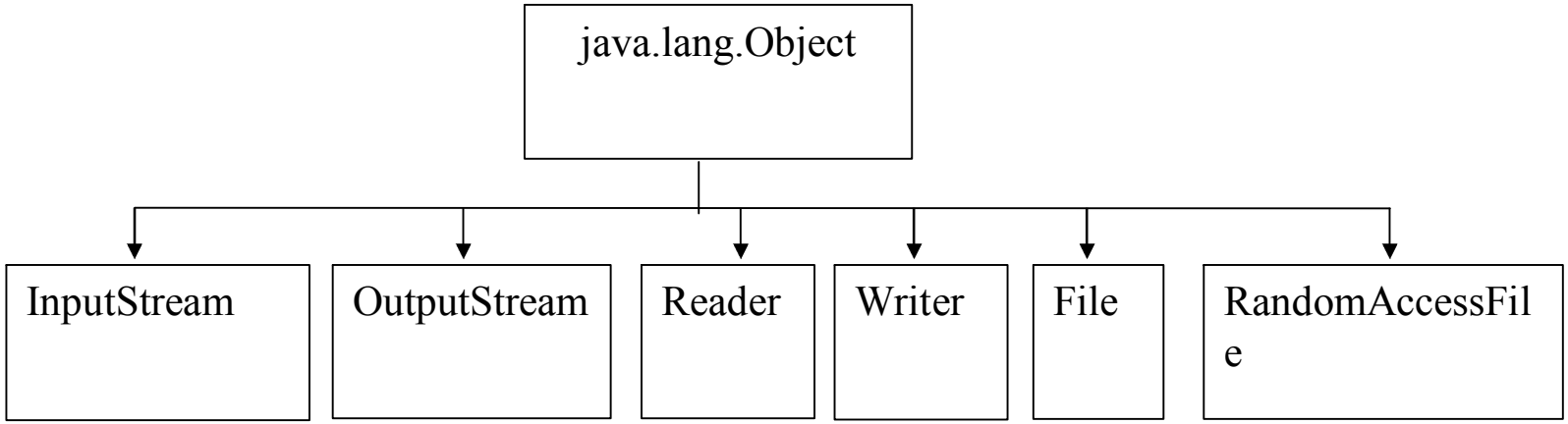
流式输入输出的特点是数据的获取和发送沿数据序列的顺序进行，即每一个数据都必须等待排在它前面的数据，等前面的数据读入或送出之后才能被读写。

8.3.2 java.io包

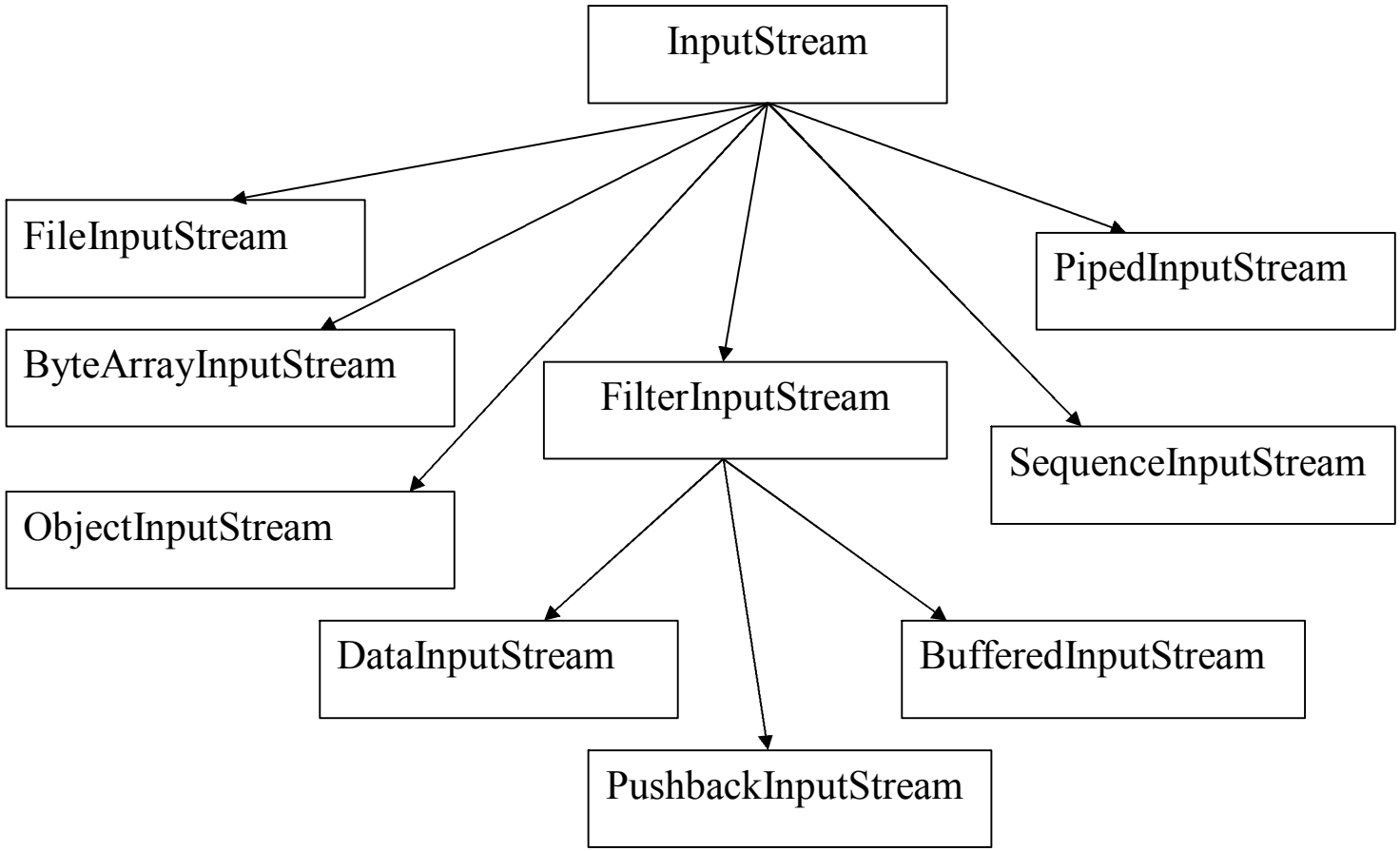
Java语言中提供了比较完善的处理输入输出数据的功能，实现这些I/O操作的类和接口都在java.io包中。java.io包中提供了各种各样的输入输出流类，它们都是Object类的直接子类，每一个流类代表一种特定的输入或输出流。

基本输入流（InputStream）和基本输出流（OutputStream）是处理以8位字节为基本单位的字节流类，读写以字节为单位进行；Reader和Writer类是专门处理16位字符流的类，读写以字符（Unicode）为单位进行。

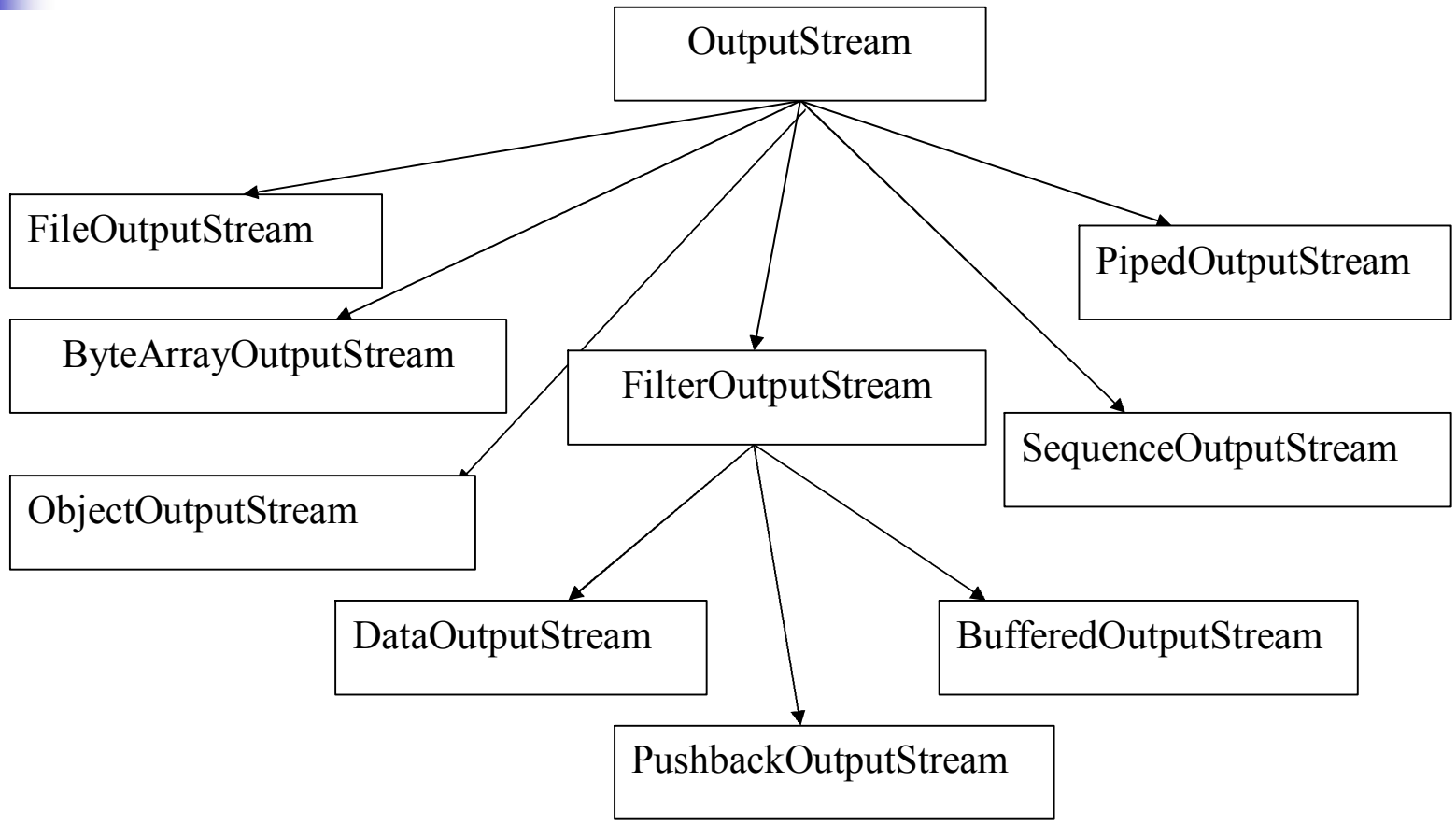
基本流类的继承关系


java.lang.Object
InputStream
OutputStream
Reader
Writer
File
RandomAccessFile

输入流的继承关系


InputStream
FileInputStream
ByteArrayInputStream
ObjectInputStream
FilterInputStream
DataInputStream
BufferedInputStream
PushbackInputStream
PipedInputStream
SequenceInputStream

续


OutputStream
FileOutputStream
ByteArrayOutputStream
FilterOutputStream
ObjectOutputStream
DataOutputStream
BufferedOutputStream
PushbackOutputStream
PipedOutputStream
SequenceOutputStream

8.3.3 InputStream与OutStream类

1. InputStream类

InputStream类是个抽象类，作为字节输入流的直接或间接的父类，它定义了许多有用的、所有子类必须的方法，包括读取、移动指针、标记、复位、关闭等方法。这些方法大多可能抛出IOException异常。

public int read( )：从输入流的当前位置读取一个字节的数据，并返回一int型值，如果当前位置没有数据则返回-1。

public int read(byte[] b)：该方法从输入流的当前位置开始读取多个字节，并将它们保存到字节数组b中，同时返回所读到的字节数，如果当前位置没有数据则返回-1。

public int read(byte[] b , int off , int len)：该方法从输入流的当前位置读取指定个数（len）的字节，并将读取的字节保存到字节数组b中，并且要从数组b指定索引（off）位置开始起，同时返回所读到的字节数，如果当前位置没有数据则返回-1。

public int available( )：返回输入流中可以读取的字节数。

public void close( )：关闭输入流，并释放流占用的系统资源。

2. OutputStream类

OutputStream类也是抽象类，作为字节输出流的直接或间接的父类，当程序需要向外部设备输出数据时，需要创建OutputStream的某一个子类的对象来完成。

public void write(int b)：将int型变量b的低字节写入到数据流的当前位置。

public void write(byte [] b)：将字节数组b的b.length个字节写入到数据流的当前位置。

public void write(byte[] b, int off , int len)：将字节数组b由下标off开始，长度为len的字节数据写到输出流。

public void flush( )：将缓冲区中的数据写到外设并清空缓冲区。

public void close( )：关闭输出流并释放输出流占用的资源。

8.3.4 具体输入输出流

由**InputStream**类和**OutputStream**类派生出来的一些常用的子类。

FileInputStream类和FileOutputStream类：负责从本地文件的读写数据。

PipedInputStream类和PipedOutputStream类：用于以管道的方式在应用程序线程间进行数据传输，一个线程的PipedInputStream对象从另一个线程的PipedOutputStream对象中读取数据。

FilterInputStream类和FilterOutputStream类：过滤输入输出流，主要能够对输入输出的数据作类型或格式上的转换，实现了对二进制字节的编码转换，而它又进一步派生出一些具体的子类，如：DataInputStream、DataOutputStream和BufferedInputStream、BufferedOutputStream等。

ByteArrayInputStream类和ByteArrayOutputStream类：用于进行内存数据的输入和输出。

8.4 File类

8.2.1 File类的构造函数

File类有三个构造函数：

① public File(String pathname)：创建一个对应于参数pathname的File类对象。参数pathname是包含目录和文件名的字符串。如果没有文件名，则代表目录。

例如：

```
File file1=new File("d:\\javapj\\myinput");
File file2=new
File("d:\\javapj\\myinput\\mysys.java");
```

续

② public File(String parent, String child)：

该构造函数将pathname分成两部分parent和child，参数parent表示目录或文件所在路径，参数child表示目录或文件名称。

例如：

```
File file1=new File("d:\\javapj" , "myinput");
File file2=new File("d:\\javapj\\myinput","mysys.java");
```

③ public File(File parent, String child)：该构造函数与上面一个的不同之处，在于将parent的参数类型由String变为File，代表parent是一个已经创建了的File类文件对象（指向目录）。

例如：

```
File file1=new File("d:\\javapj\\myinput");
File file2=new File(file1, "mysys.java");
```

8.4.2 File类的常用方法

public boolean canWrite( )：返回文件是否可写。
public boolean canRead( )：返回文件是否可读。
public boolean createNewFile( )：当文件不存在时创建文件。
public boolean delete( )：从文件系统内删除该文件。
public void deleteOnExit( )：程序顺利结束时从系统中删除文件。
public boolean exists( )：判断文件是否存在。
public File getAbsoluteFile( )：以**File**类对象形式返回文件的绝对路径。
public String getAbsolutePath( )：以字符串形式返回文件的绝对路径。
public String getName( )：以字符串形式返回文件名称。
public String getParent( )：以字符串形式返回文件父目录路径。
public String getPath( )：以字符串形式返回文件的相对路径。
public File getParentFile( )：以**File**类对象形式返回文件父目录的路径。
public boolea isFile()：判断该**File**对象所对应的是否是文件。
public long lastModified() ：返回文件的最后修改时间。
public int length()：返回文件长度。
public boolean mkdir()：在当前目录下生成指定的目录。
public boolean setReadOnly()：将文件设置为只读。
public String toString()：将文件对象的路径转换为字符串返回。

例8.7 使用File类管理文件和目录。

```
import java.io.*;
import java.util.Date;
public class filetest {
public static void main(String []args)
{String filename="d:\\javapj\\ex.java";
File myfile=new File(filename);
if(!myfile.exists() )
{   System.err.println(filename+"未找到!");
    return;  }
if( myfile.isDirectory() )
{ System.err.println("文件对象"+myfile.getName()+"是
    目录!");
```

续

```
        File ds=new File("mydata");
        if(!ds.exists())
{       ds.mkdir();
        System.out.println("目录"+ds.getAbsolutePath()+"创建结束!");      }
        return;             }
if(myfile.isFile())
{       System.out.println("文件对象:"+myfile.getAbsolutePath());
        System.out.println("文件字节数:"+myfile.length());
        System.out.println("文件是否能读:"+myfile.canRead());
        if(myfile.canWrite()){
                System.out.println("设置文件为只读:"+myfile.setReadOnly());}
                System.out.println("文件是否可写:"+myfile.canWrite());
                Date fd=new Date(myfile.lastModified());
                System.out.println("文件上次修改时间:"+fd.toString());
  }}}
```

例8.7程序运行结果如图所示

8.5 文件输入与输出

8.5.1 FileInputStream类和FileOutputStream类的使用

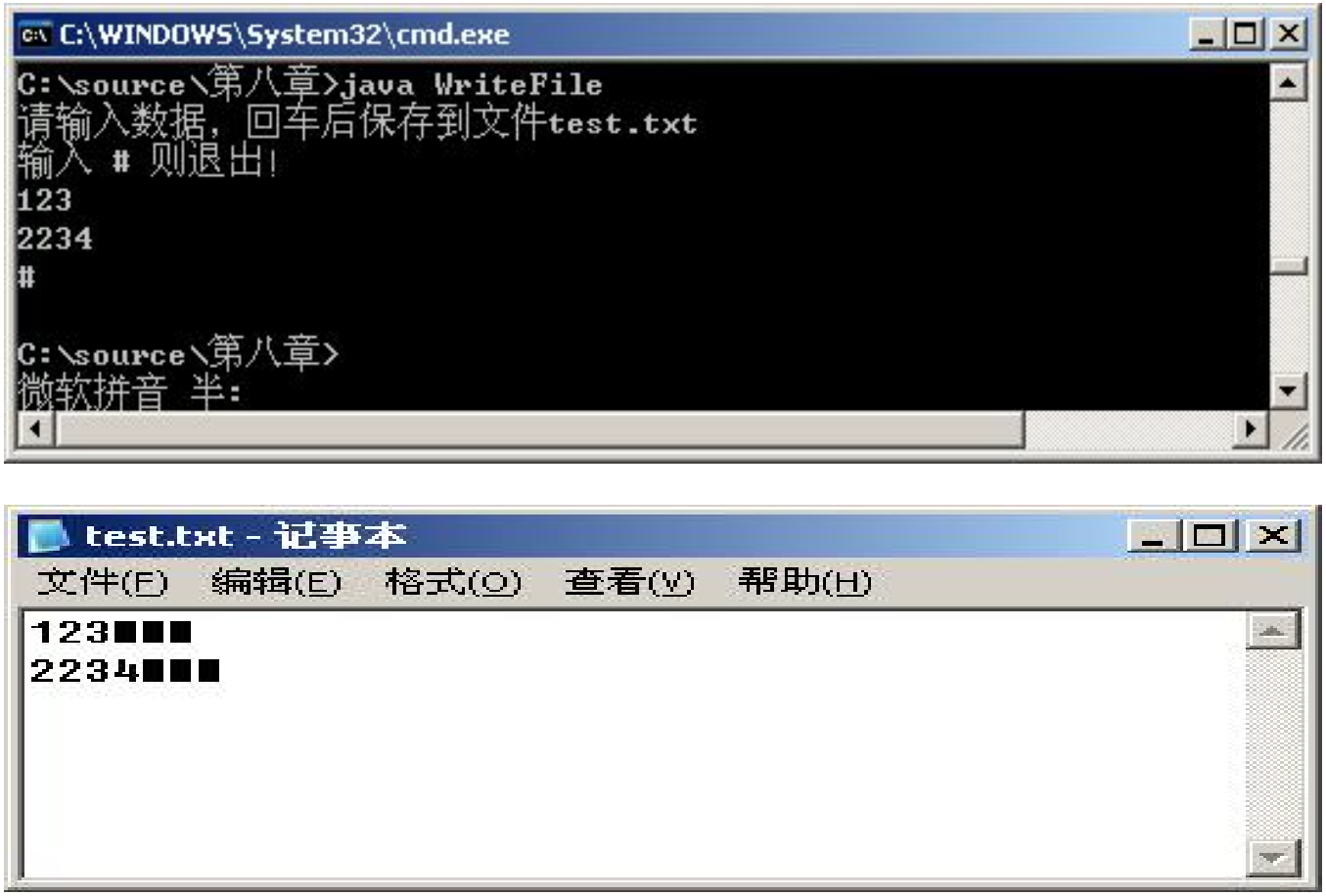
FileInputStream类和FileOutputStream类的构造函数是创建一个输入输出的对象，通过引用该对象的读写方法，来完成对文件的输入输出操作。在构造函数中，需要指定与所创建的输入输出对象相连接的文件。当然，要构造一个FileInputStream对象，所连接的文件必须存在而且是可读的；构造一个FileOutputStream对象如果输出文件已经存在且可写，该文件内容会被新的输出所覆盖。

例 8.8

编写程序，接收用户从键盘输入的数据，回车后保存到文件test.txt中。若用户输入符号#，则退出程序。

```
import java.io.*;
public class WriteFile{
    public static void main(String args[])
    {       byte buffer[]=new byte[128];
            System.out.println("请输入数据，回车后保存到文件test.txt");
            System.out.println("输入 # 则退出！ ");
            try{
                    FileOutputStream f=new FileOutputStream("test.txt");
                     while(true){
                             int n=System.in.read(buffer);
                             if(buffer[0]=='#' )   break;
                             f.write(buffer,0,n);
                             f.write('\n');
                    }
                    f.close();
            }catch(IOException e)
                    {System.out.println(e.toString());
    }}}
```

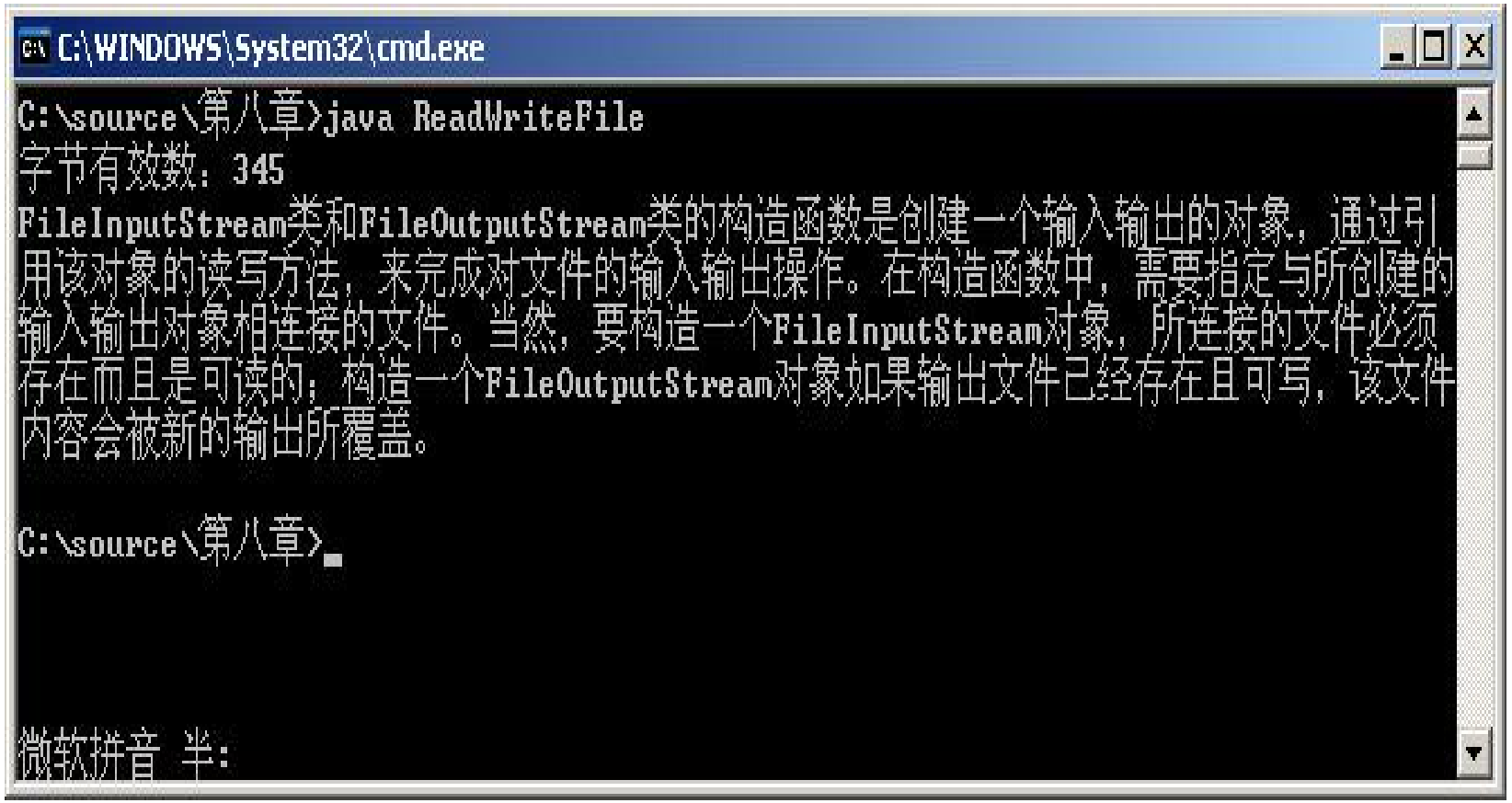
例8.8程序运行结果如图所示



例 8.9 使用FileInputStream类与FileOutputStream类复制文件。

```
import java.io.*;
class ReadWriteFile{
      public static void main(String[] args) {
        String file1,file2 ;
        int ch = 0 ;
        file1 = "readme.txt" ;
        file2="readme.bak";
        try  {
                        FileInputStream fis = new FileInputStream(file1);
                        FileOutputStream fos=new FileOutputStream(file2);
                        int size=fis.available();
                        System.out.println("字节有效数："+size);
              while ((ch=fis.read())!=-1){
              System.out.write(ch);
              fos.write(ch);
              }
                        fis.close();
                        fos.close();
        }
catch (IOException e){
                        System.out.println(e.toString());
              } }}
```

例8.9程序运行结果如图所示



8.5.2 读写文件中的基本数据类型

DataInputStream和DataOutputStream分别实现了java.io包中的DataInput和DataOutput接口，能够读写Java基本数据类型的数据和Unicode编码格式的字符串。这样，在输入输出数据时就不必关心该数据究竟包含几个字节了。

DataInputStream类和DataOutputStream类是从过滤流类继承过来，这两个流的对象均不能独立地实现数据的输入和输出处理，必须与其他输入流与输出流对象一起使用，才能实现不同类型数据的读写。

续

DataInputStream 类的读方法	DataOutputStream 类的写方法
boolean readBoolean()	void writeBoolean(Boolean v)
byte readByte()	void writeByte(int v)
char readChar()	void writeBytes(String s)
double readDouble()	void writeChar(int v)
float readFloat()	void writeChars(String s)
int readInt()	void writeDouble(double v)
long readLong()	void writeFloat(float v)
short readShort()	void writeInt(int v)
int readUnsignedByte()	void writeLong(int v)
int readUnsignedShort()	void writeShort(int v)
void readFully(byte[b)	void writeUTF(String str)
void readFully(byte[b, int off,int len)	
int skipBytes(int n)	
String readUTF()	

例8.10

使用DataInputStream类和DataOutputStream类读写格式文件。

```
import java.io.*;
public class fdsRW{
 public static void main(String[] args){
     String file="student.dat";
     Student s1=new Student();
     Student s2=new Student(10,"张飞",16,'A',true);
     try{
          FileOutputStream fo=new FileOutputStream(file);
     //创建文件输出流对象
          DataOutputStream out=new DataOutputStream(fo); //创建数据输
     出流对象
          out.writeInt(s1.sno);          //写文件
          out.writeUTF(s1.name);
          out.writeInt(s1.age);
          out.writeChar(s1.grade);
          out.writeBoolean(s1.sex);
```

续1

```
out.writeInt(s2.sno);
out.writeUTF(s2.name);
out.writeInt(s2.age);
out.writeChar(s2.grade);
out.writeBoolean(s2.sex);
out.close();            //关闭数据输出流
fo.close();//关闭文件输出流
System.out.println("文件:"+file+"创建完毕!");
System.out.println("开始读取文件内容:");
FileInputStream fi=new FileInputStream(file);                    //创建文件输入流对象
DataInputStream in=new DataInputStream(fi);     //创建数据输入流对象
```

续2

```
for(int i=1;i<=2;i++){//读取文件内容
            int sno=in.readInt();
            String sname=in.readUTF();
            int age=in.readInt();
            char grade=in.readChar();
            boolean sex=in.readBoolean();

            System.out.println(sno+"\t"+sname+"\t"+age+"\t"+grade
    +"\t"+sex);
            }
            in.close();
            fi.close();
    }catch(IOException e){
             System.out.println(e.toString());          }
}}
```

续3

```
class Student{        int sno;
        String name;
        int age;
        char grade;
        boolean sex;
        public Student(){
                this.sno=0;
                this.name="未知";
                this.age=0;
                this.grade='C';
                this.sex=true;
        }
        public Student(int sno,String name,int age,char grade,boolean sex){
                if(sno>0) this.sno=sno;
                this.name=name;
                this.age=age;
                this.grade=grade;
                this.sex=sex;
        }}
```

例8.10程序运行结果如图所示

8.5.3 随机文件的读取

完成随机文件读写的类与Java中大部分的输入输出流类不同，没有针对输入、输出对应出现两个相应的类，它是由java.lang.Object直接继承而来的，仅使用一个RandomAccessFile类来完成。所以RandomAccessFile类的对象既可以用于数据输入，也可以用于数据输出。

1．RandomAccessFile的构造函数

RandomAccessFile类提供了两个构造函数：

```
public RandomAccessFile(File file,String mode);
public RandomAccessFile(String
    name,String mode);
```

2．RandomAccessFile类指针控制方法

public long getFilePointer()：获取当前指针指向文件的位置。

pulbic void seek(long pos)：　　将指针移动到参数pos指定的位置。

public int skipBytes(int n)：指针从当前位置向后移动n个字节位置，并返回指针实际移动的字节数。

3．RandomAccessFile类读写数据的常用方法

public void close()：关闭输入输出流；

public long length()：获取文件的长度。

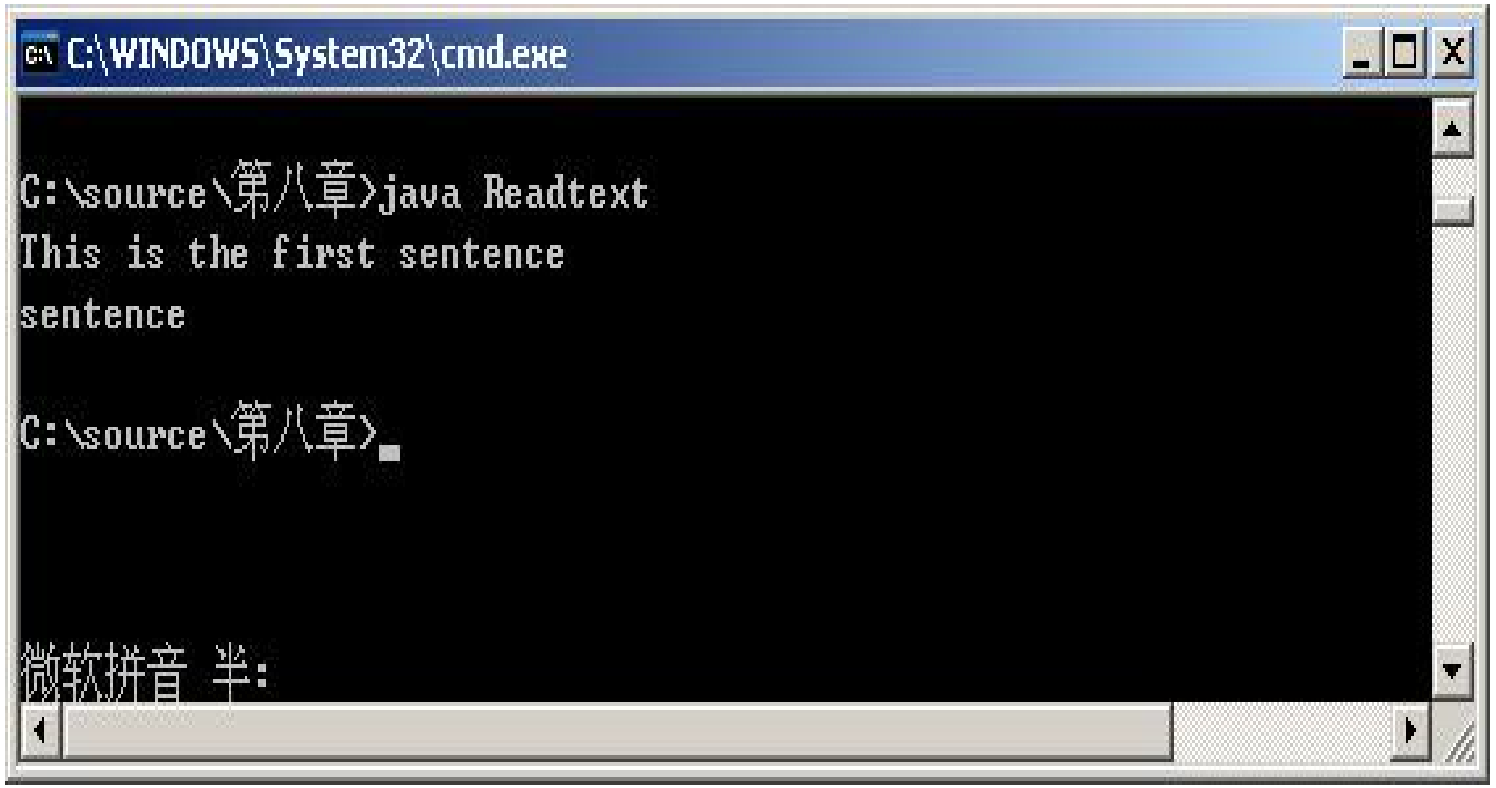
例8.11　随机读取文件中的字符信息。

```
import java.io.*;
public class Readtext
{
        public static void main(String[] args)
        {       String str1;
                char ch1;
                int n;
                try{
                        File mytxt=new File("read.txt");
                        RandomAccessFile ra=new
    RandomAccessFile(mytxt,"rw");
```

续

```
ra.write("This is the first sentence".getBytes());
                                ra.writeChar('\n');
                                ra.write("Java Program
    Design".getBytes());
                                ra.writeChar('\n');
                                ra.seek(0);
                                str1=ra.readLine();
                                System.out.println(str1);
                                ra.seek(18);
                                System.out.println(ra.readLine());
                                ra.close()
                        }
catch(IOException e)    {
                                System.out.println(e.toString());
    }}}
```

例8.11程序运行结果如图所示



例8.12 利用RandomAccessFile实现记录式访问。

```
import java.io.*;
public class RaFile
{   public static void main(String[] args)
        {
            Student s[]=new Student[4];
            s[0]=new Student("zhangsan",17,false);
            s[1]=new Student("lisi",18,true);
            s[2]=new Student("wangwu",20,true);
            s[3]=new Student("zhaoliu",19,false);
            try{
                RandomAccessFile ra=new
    RandomAccessFile("student.dat","rw");
```

续1

```
for(int i=0;i<4;i++) //将4个学生信息写入文件
                    {
            ra.writeBytes(s[i].name); //将字符串按字节写入
            ra.writeInt(s[i].age);
            ra.writeBoolean(s[i].sex);
                                }
            System.out.println("随机文件字节数："+ra.length());
            ra.seek(0);          //文件指针指向开始位置
            System.out.println("第一条学生记录：");
            byte b[]=new byte[8];
            ra.read(b);
            int age=ra.readInt();
            boolean sex=ra.readBoolean();
            System.out.println(new String(b)+"\t"+age+'\t'+sex);
            ra.skipBytes(26); //访问第四个记录
```

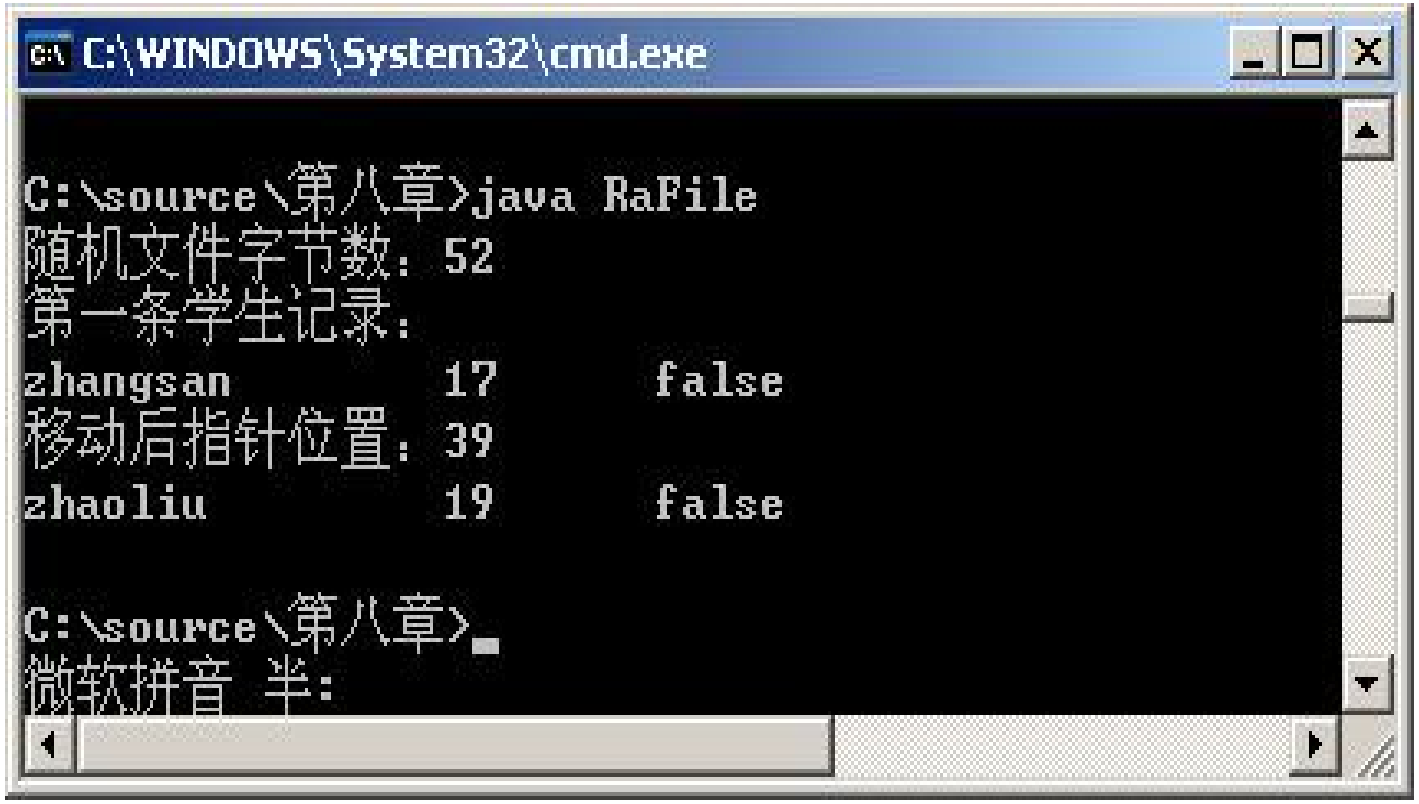
续2

```
System.out.println("移动后指针位置："+ra.getFilePointer());
                        ra.read(b);
                        age=ra.readInt();
                        sex=ra.readBoolean();
                        System.out.println(new
    String(b)+"\t"+age+'\t'+sex);
                        ra.close();
                    }
                    catch(IOException e){
                        System.out.println(e.toString());
}}}
```

续3

```
class Student{     //学生类的定义
        String name;
        int age;
        boolean sex;
        final static int LEN=8;
        Student(String name,int age,boolean sex){
                if(name.length()>LEN){
                name = name.substring(0,8);        //超过8个字符时，
    取前8个
                }
                else{while(name.length()<LEN)
                name=name+"\u0000";    //不足8位，补足
                }
                this.name=name;
                this.age=age;
                this.sex=sex;
        }}
```

例8.12程序运行结果如图所示



8.6 标准输入和输出

8.6.1 System.in对象

System.in是InputStream类的对象，调用System.in.read()方法就可以实现标准输入的读操作。

8.6.2 System.out对象

System.out是PrintStream类的对象，PrintStream类是FilterOutputStream类的子类，其中定义了可输出多种不同类型数据的方法print()和println()方法。

例 8.13　标准输入输出示例。

```
import java.io.*;
public class StandardIO1{
    public static void main(String []args)
    {
        int ch;
        System.out.println("请输入一行字符");
        try{
            while((ch=System.in.read())!='\r')
                System.out.write(ch);
        }catch(IOException e){
            System.out.println(e.toString());
        }
        System.out.write('\n');
    }
}
```

例8.13程序运行结果如图所示

8.6.3 数据类型的转换

1. 基本数据类型变量和包装类对象之间的相互转换

简单类型的变量转换为相应的包装类对象，可以利用包装类的构造函数。

例如以下代码将产生一个包装类Float的实例floatInstance，

```
float fNumber=5.5f;
Float floatInstance=new Float( fNumber);
```

在各个包装类中，总有形为××Value()的方法，来得到其对应的基本数据类型。

例如：Double dInstance=new Double(123.456d);

```
double x=dInstance.doubleValue();
int y=dInstance.intValue();
```

2. 字符串和其他数据类型的相互转换

各个包装类都定义了可以把数值字符串转换为对应的基本数据类型的静态方法，形如parse××(String str)。以下代码实现了字符串到基本数据类型的转换：

```
int i=Integer.parseInt("123");
long l=Long.parseLong("123");
float f=Float.parseFloat("123.45");
double d=Double.parseDouble("123.4567d");
```

续

所有的包装类都存在一个名为toString()的方法可以将其转换成对应的字符串，而对于整型类和长整型类，还可以使用toBinaryString(int i)、toHexString(int i)、toOctalString(int i)分别以二进制、十六进制和八进制形式的字符串返回。

例如：Integer xInteger=new Integer(500);

String str=xInteger.toString( );

例8.14 求从键盘输入的n个数的平均数。

```
import java.io.*;
public class average {
    public static void main(String []args){
        byte b[]=new byte[10];
        String str;
        int n=0;
        double sum=0,d,avg;
        System.out.print("要对几个数求平均?");
        try{    System.in.read(b);
                str=new String(b).trim();   //去除多余的/r、/n、
    空格符
                n=Integer.parseInt(str);}
catch(IOException e){System.out.println(e.toString());}
```

续

```
catch(NumberFormatException e){
        System.out.println("请不要输入0-9以外的其他字符！");
        System.exit(-1);}    //程序异常结束
        for(int i=1;i<=n;i++){
                System.out.print(i+": ");
                try{        System.in.read(b);
                        str=new String(b);
                        Double x=new Double(str);
                        d=x.doubleValue();
                        sum+=d;}
catch(IOException e){System.out.println(e.toString());        }
catch(NumberFormatException e){        i--;        //重新输入
                System.out.println("请正确输入实数！");}}
        System.out.println("平均值="+sum/n);}}
```

例8.15程序运行结果如图所示