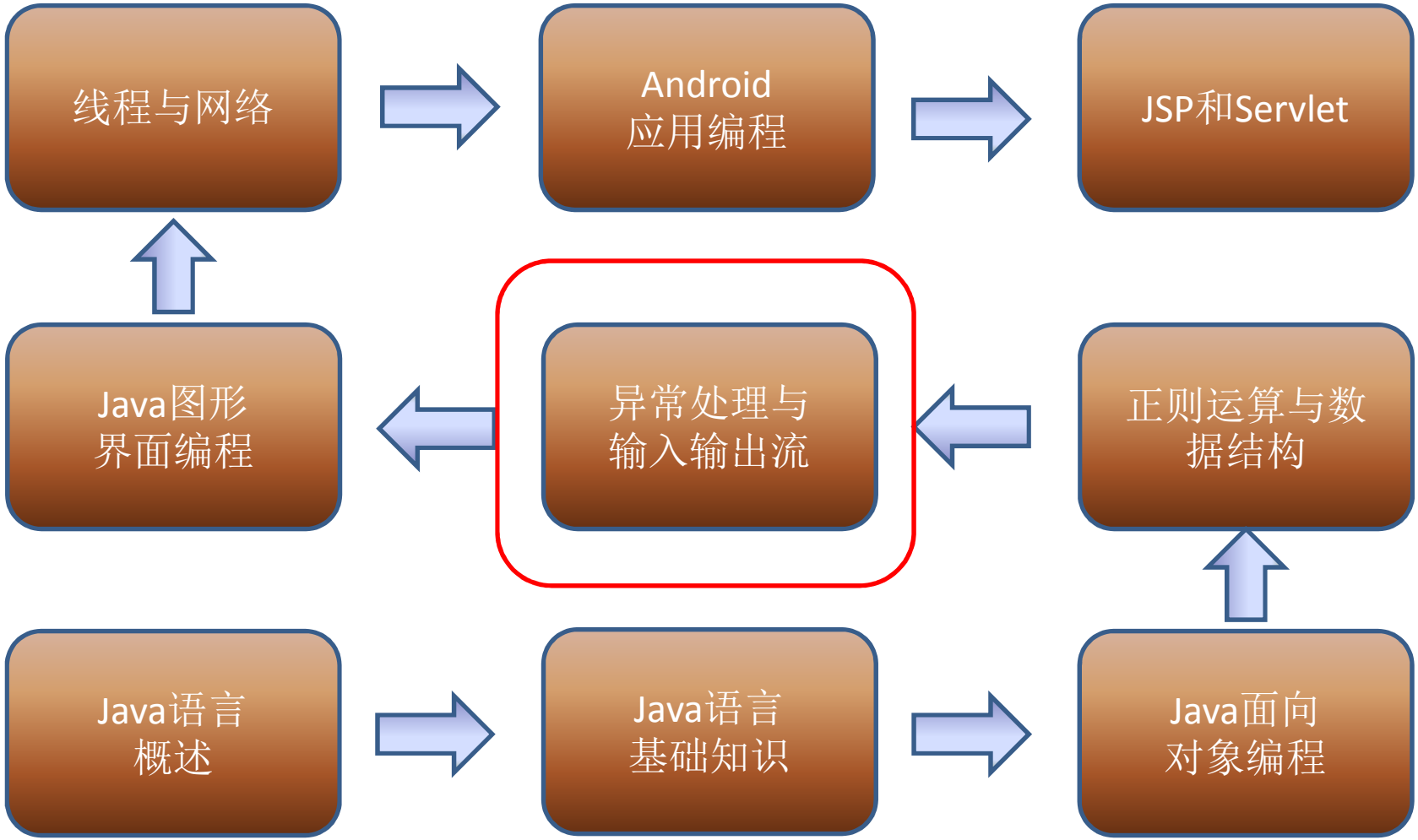
第五讲 Java的异常处理和输入输出流

课程内容安排


线程与网络
Android
应用编程
JSP和Servlet
Java图形
界面编程
异常处理与
输入输出流
正则运算与数
据结构
Java语言
概述
Java语言
基础知识
Java面向
对象编程

课前思考

1. 打开文件时，文件路径不对怎么办？
2. 如何读取硬盘上的某个文件内容？
3. 如何知道文件系统中有哪些目录和子目录？
4. 如何往硬盘上写文件？
5. 如何接收键盘输入？
6. 如何接收来自网络上的传输内容？

异常处理
Exception

什么是异常

- 异常就是在程序的运行过程中所发生的异常事件，它中断指令的正常执行。

异常示例


```
import java.io.*;
class ExceptionDemo1{
   public static void main( String args[ ] ){
      FileInputStream fis = new FileInputStream( "text" );
      int b;
      while( (b=fis.read())!=-1 ){
            System.out.print( b );
      }
      fis.close( );
   }
}
```

异常示例

C:\>*javac ExceptionDemo1.java*

ExceptionDemo1.java:6: Exception java.io.FileNotFoundException must be caught, or it must be declared in the throws clause of this method.

FileInputStream fis = new FileInputStream("text");
^

ExceptionDemo1.java:8: Exception java.io.IOException must be caught, or it must be declared in the throws clause of this method.

while((b=fis.read())!=-1){
^

2 errors

异常示例


```
class ExceptionDemo2{
  public static void main( String args[ ] ){
    int a = 0;
    System.out.println( 5/a );
  }
}
```

异常示例

C:\>*javac ExceptionDemo2.java*

C:\>*java ExceptionDemo2*

java.lang.ArithmeticException: /by zero at ExceptionDemo2.main(ExceptionDemo2.java:4)

异常处理机制

- 在Java程序的执行过程中，如果出现了异常事件，就会生成一个异常对象。
- 生成的异常对象将传递给Java运行时系统，这一异常的产生和提交过程称为抛出(throw)异常。

异常处理机制

- 当Java运行时系统得到一个异常对象时，它将会寻找处理这一异常的代码。找到能够处理这种类型的异常的方法后，运行时系统把当前异常对象交给这个方法进行处理，这一过程称为捕获(catch)异常。
- 如果Java运行时系统找不到可以捕获异常的方法，则运行时系统将终止，相应的Java程序也将退出。

异常(Throwable)分类

- Error

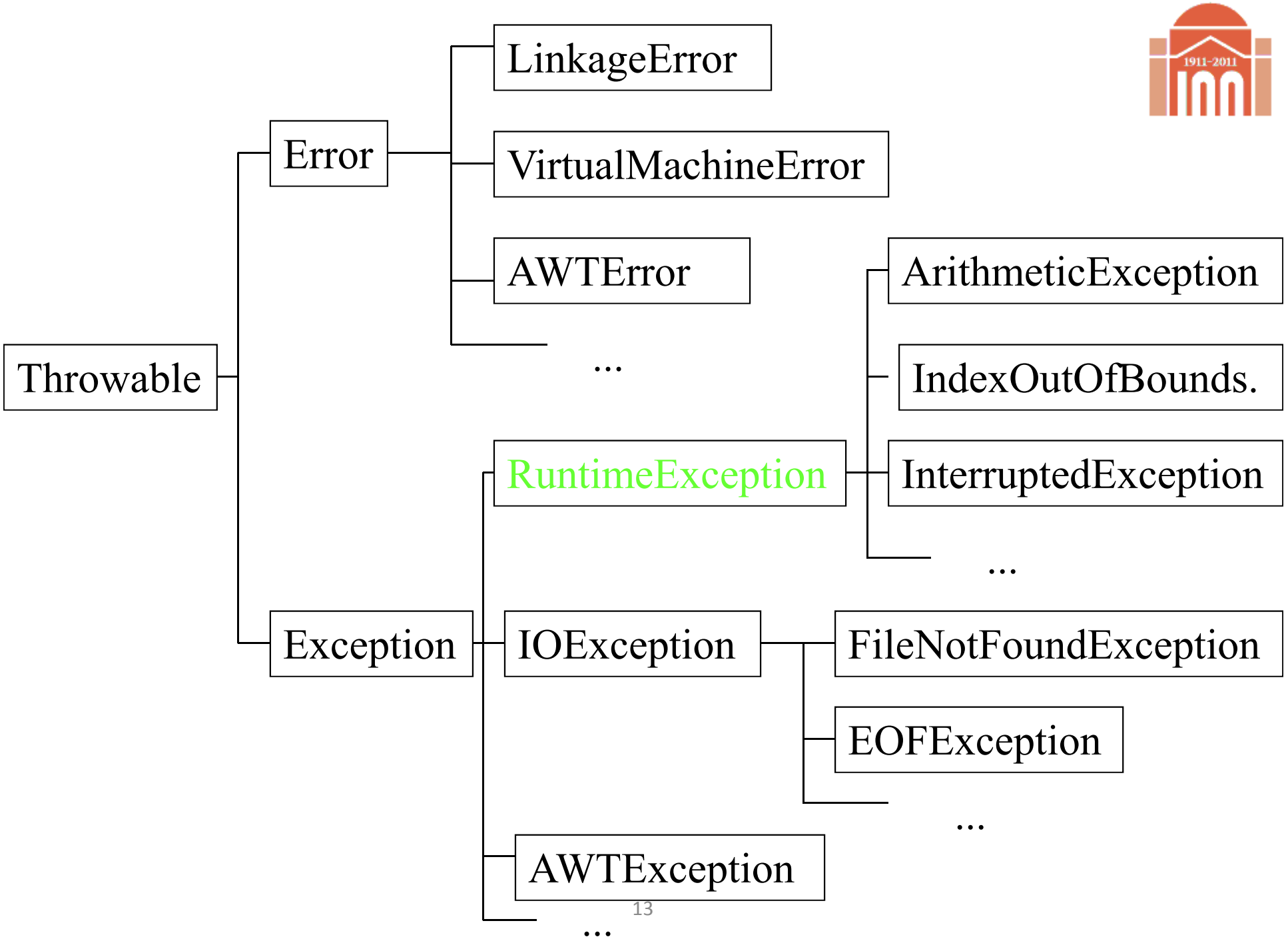
  动态链接失败，虚拟机错误等，通常Java程序不应该捕获这类异常，也不会抛出这种异常。

- Exception
  - 运行时异常

    继承于RuntimeException。Java编译器允许程序不对它们做出处理。

  - 非运行时异常

    除了运行时异常之外的其他由Exception继承来的异常类。Java编译器要求程序必须捕获或者声明抛出这种异常。


Throwable
Error
LinkageError
VirtualMachineError
AWTError
...
Exception
RuntimeException
ArithmeticException
IndexOutOfBounds.
InterruptedException
...
IOException
FileNotFoundException
EOFException
...
AWTException
...

- 捕获异常

  捕获异常是通过try-catch-finally语句实现的。

```
try{
……
  }catch( ExceptionName1 e ){
  ……
  }catch( ExceptionName2 e ){
  ……
  }
  ……
  }finally{
  ……
}
```

- try

  捕获异常的第一步是用try{...}选定捕获异常的范围，由try所限定的代码块中的语句在执行过程中可能会生成异常对象并抛出。

- catch

每个try代码块可以伴随一个或多个catch语句，用于处理try代码块中所生成的异常事件。catch语句只需要一个形式参数指明它所能够捕获的异常类型,这个类必须是Throwable的子类,运行时系统通过参数值把被抛出的异常对象传递给catch块.

在catch块中是对异常对象进行处理的代码，与访问其它对象一样，可以访问一个异常对象的变量或调用它的方法。getMessage( )是类Throwable所提供的方法，用来得到有关异常事件的信息，类Throwable还提供了方法printStackTrace( )用来跟踪异常事件发生时执行堆栈的内容。


```
try{
      ......
  }catch( FileNotFoundException e ){
      System.out.println( e );
      System.out.println("message:
"+e.getMessage() );
      e.printStackTrace( System.out );
  }catch( IOException e ){
      System.out.println( e );
  }
```

catch语句的顺序

- 捕获异常的顺序和不同catch语句的顺序有关,当捕获到一个异常时,剩下的catch语句就不再进行匹配。因此，在安排catch语句的顺序时，首先应该捕获最特殊的异常，然后再逐渐一般化。也就是一般先安排子类，再安排父类。

- finally

  捕获异常的最后一步是通过finally语句为异常处理提供一个统一的出口，使得在控制流转到程序的其它部分以前，能够对程序的状态作统一的管理。不论在try代码块中是否发生了异常事件，finally块中的语句都会被执行。

声明抛出异常

- 如果在一个方法中生成了一个异常，但是这一方法并不确切地知道该如何对这一异常事件进行处理，这时，一个方法就应该声明抛出异常，使得异常对象可以从调用栈向后传播，直到有合适的方法捕获它为止。

声明抛出异常

- 声明抛出异常是在一个方法声明中的throws子句中指明的。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，而是声明抛出它们


```
public class ThrowsException {
    static void proc( int sel )  throws
    ArithmeticException,ArrayIndexOutOfBoundsException{
System.out.println("In Situation"+sel);
if( sel==0 ){
                System.out.println("no Exception caught");
                return;
      }else if( sel==1 ){
                int iArray[]=new int[4];
                iArray[10]=3;
      }
    }
```


```
    public static void main( String args[] ){
      try{
                proc( 0 );
                proc( 1 );
      }catch( ArrayIndexOutOfBoundsException e ){
                System.out.println("Catch "+e);
      }finally{
                System.out.println("in Proc finally");
      }
    }
}
```

运行结果

In Situation0

no Exception caught

In Situation1

Catch java.lang.ArrayIndexOutOfBoundsException: 10

in Proc finally

抛出异常

- 抛出异常首先要生成异常对象，异常或者由虚拟机生成，或者由某些类的实例生成，也可以在程序中生成。生成异常对象是通过throw语句实现的。

```
IOException e=new IOException();
throw e ;
```

可以抛出的异常必须是Throwable或其子类的实例。下面的语句在编译时将会产生语法错误：

```
throw new String("want to throw");
```

异常类的使用

- FileInputStream 的API

public FileInputStream(String name)

throws FileNotFoundException

异常示例


```
import java.io.*;
class ExceptionDemo1{
  public static void main( String args[ ] ){
    FileInputStream fis = new
  FileInputStream( "text" );
……
  }
}
```

异常类的使用

- 积极处理方式:

```
import java.io.*;
class ExceptionDemo1{
   public static void main( String args[ ] ){
try{   FileInputStream fis = new
                FileInputStream( "text" );
     } catch(FileNotFoundException e) {
                      ……}
  ……
   }
}
```

异常类的使用

- 消极处理方式:

```
import java.io.*;
class ExceptionDemo1{
   public static void main( String args[ ] )
            throws FileNotFoundException{
   FileInputStream fis = new
            FileInputStream( "text" );
……
   }
}
```

异常类的使用

- 如果采用消极处理方式,则由调用该方法的方法进行处理;但是调用该方法的方法也可以采用消极和积极两种处理方式,一直传递到Java运行环境.


```
class ThrowDemo {
    static void demoproc () {
        try {
            throw new NullPointerException("demo");
        } catch ( NullPointerException e ) {
            System.out.println("caught inside demoproc");
            throw e;
        }
    }
    public static void main ( String args[] ) {
        try {
            demoproc();
        } catch ( NullPointerException e ) {
            System.out.println("recaught: " + e );
        }
    }
}
```

运行结果

caught inside demoproc

Recaught: java.lang.NullPointerException:demo

自定义异常类

- 自定义异常类必须是Throwable的直接或间接子类
- 一个方法所声明抛出的异常是作为这个方法与外界交互的一部分而存在的。方法的调用者必须了解这些异常，并确定如何正确的处理他们。


```
class MyException extends Exception {
    private int exceptNumber;
    MyException ( int a ) {
        exceptNumber = a;
    }
    public String toString () {
        return "MyException[" + exceptNumber + "]";
    }
}
```


```
class ExceptionExample {
    static void makeExcept(int a) throws MyException {
        System.out.println("called makeExcept(" + a + ").");
        if ( a == 0 )
            throw new MyException(a);
        System.out.println("exit without exception");
    }
    public static void main ( String args[] ) {
        try {
            makeExcept(5);
            makeExcept(0);
        } catch ( MyException e ) {
            System.out.println("caught " + e );
        }
    }
}
```

运行结果

called makeExcept(5)

exit without exception

called makeExcept(0)

caught MyException[0]

异常类的使用

- 运行时异常则表示由运行时系统所检测到的程序设计问题或者API的使用不当问题,它可能在程序的任何地方出现:

(1)对于运行时异常，如果不能预测它何时发生，程序可以不做处理，而是让Java虚拟机去处理它。

(2)如果程序可以预知运行时异常可能发生的地点和时间，则应该在程序中进行处理，而不应简单的把它交给运行时系统。

异常类的使用

(3)在自定义异常类时，如果它所对应的异常事件通常总是在运行时产生的，而且不容易预测它将在何时、何处发生，则可以把它定义为运行时异常，否则应定义为非运行时异常。

输入/输出处理

java.io包

- 字节流

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

- 字符流

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

流

- 在计算机中，“流”的概念是1984年由C语言第一次引入的。“流”可以看作是一个流动的数据缓冲区，数据从数据源流向数据目的地。

字节流

- InputStream、OutputStream
- FileInputStream、FileOutputStream
- PipedInputStream、PipedOutputStream
- ByteArrayInputStream、ByteArrayOutputStream
- FilterInputStream、FilterOutputStream
- DataInputStream、DataOutputStream
- BufferedInputStream、BufferedOutputStream

字符流

- Reader、Writer
- InputStreamReader、OutputStreamWriter
- FileReader、FileWriter
- CharArrayReader、CharArrayWriter
- PipedReader、PipedWriter
- FilterReader、FilterWriter
- BufferedReader、BufferedWriter
- StringReader、StringWriter

对象流

- ObjectInputStream、ObjectOutputStream

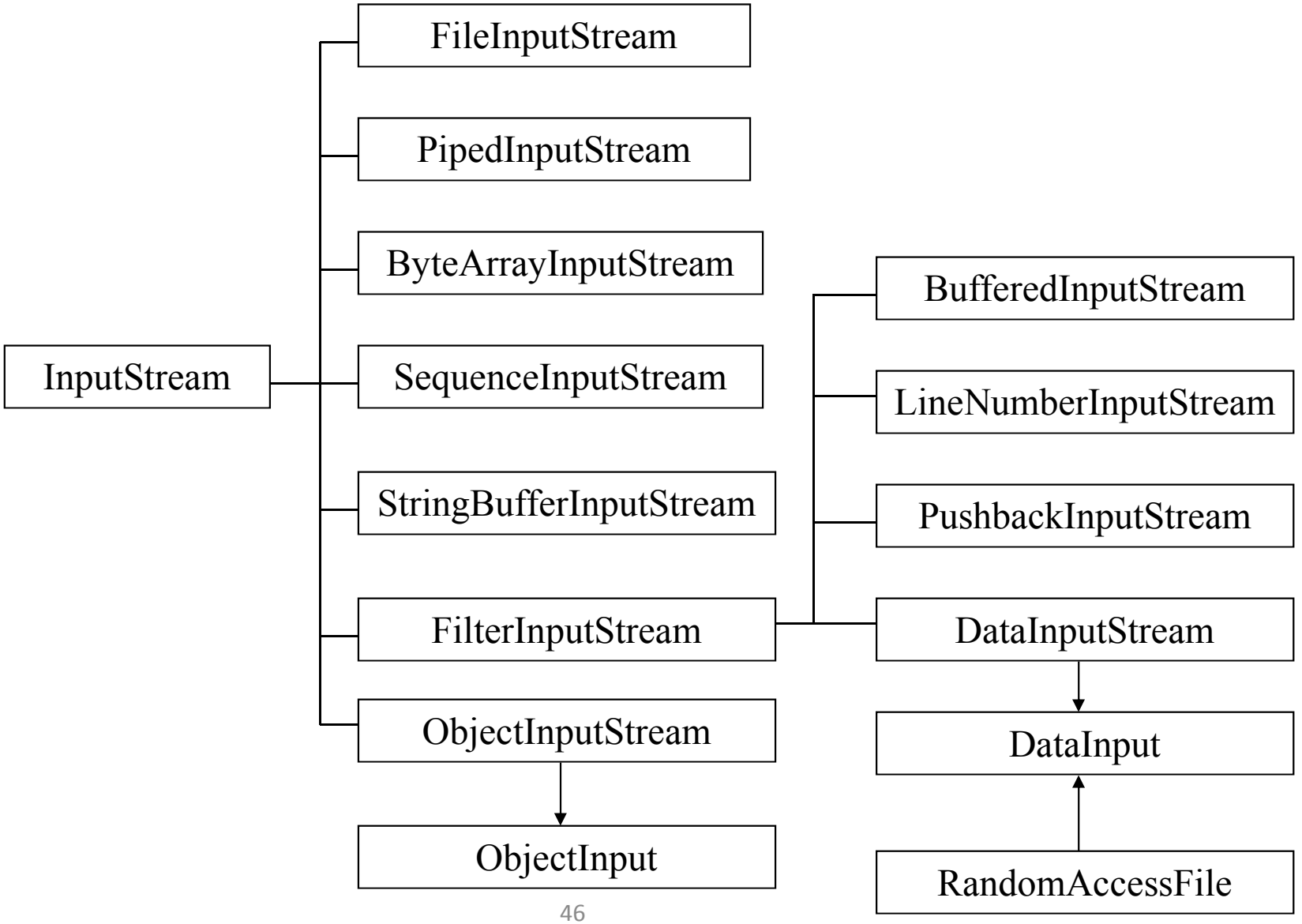
其它

- 文件处理

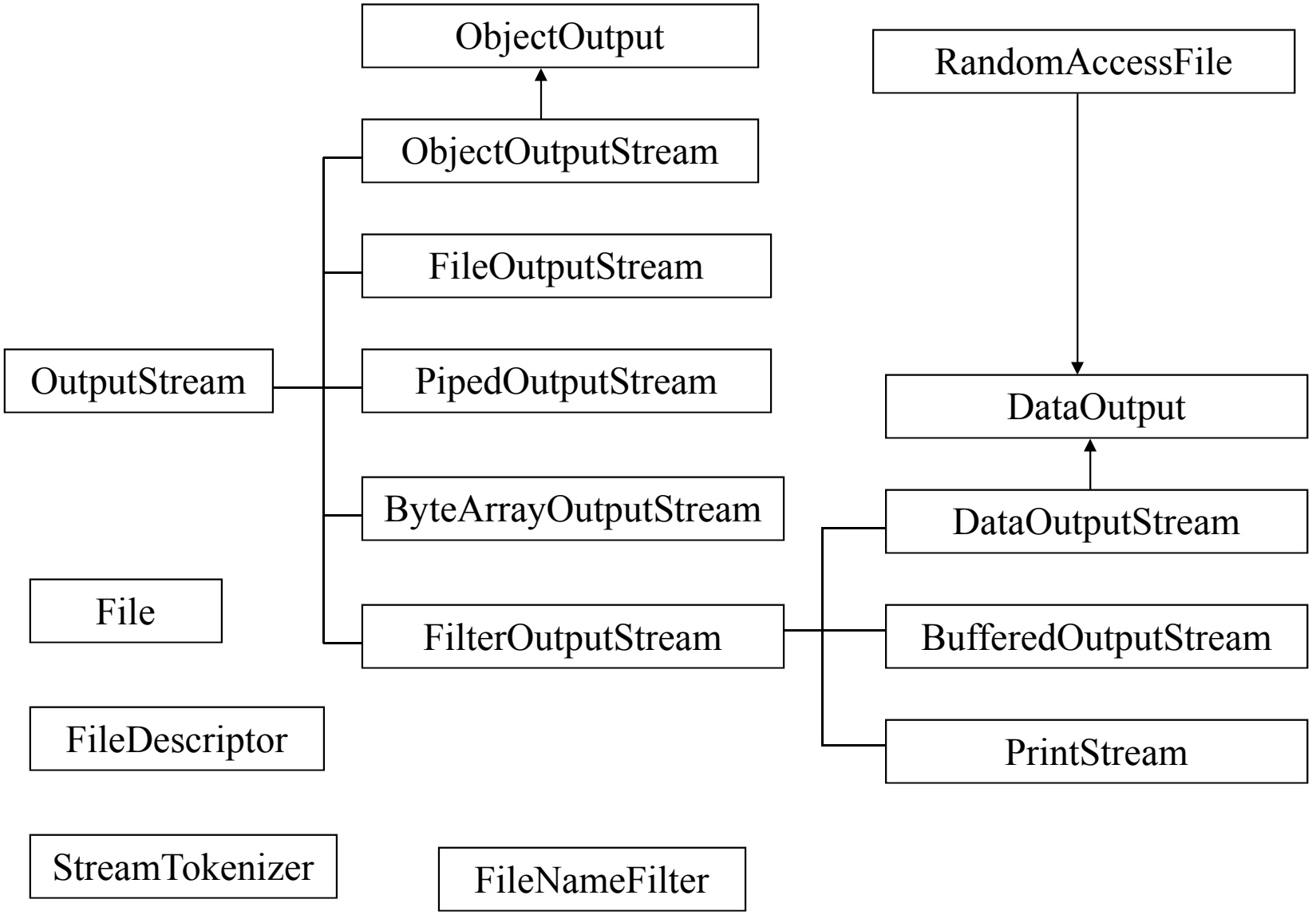
  File、RandomAccessFile；
- 接口

  DataInput、DataOutput、ObjectInput、ObjectOutput；

I/O处理的类层次


InputStream
FileInputStream
PipedInputStream
ByteArrayInputStream
SequenceInputStream
StringBufferInputStream
FilterInputStream
ObjectInputStream
ObjectInput
BufferedInputStream
LineNumberInputStream
PushbackInputStream
DataInputStream
DataInput
RandomAccessFile

I/O处理的类层次


ObjectOutput
RandomAccessFile
ObjectOutputStream
FileOutputStream
OutputStream
PipedOutputStream
DataOutput
ByteArrayOutputStream
DataOutputStream
File
FilterOutputStream
BufferedOutputStream
FileDescriptor
PrintStream
StreamTokenizer
FileNameFilter

InputStream

- 从流中读取数据

```
int read( );
int read( byte b[ ] );
int read( byte b[ ], int off, int len );
int available( );
long skip( long n )；
```

InputStream

- 关闭流

  close( );

- 使用输入流中的标记

  void mark(int readlimit);

  void reset( );

  boolean markSupported( );

OutputStream

- 输出数据

  void write(int b);

  void write(byte b[ ]);

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

- flush( )

  刷空输出流，并输出所有被缓存的字节。

- 关闭流

  close( );

文件处理

- File、FileInputStream、FileOutputStream、RamdomAccessFile

文件描述

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

  public File(String path)；

  public File(String path,String name)；

  public File(File dir,String name)；

文件描述

- 文件名的处理

String getName( );/*得到一个文件的名称（不包括路径）*/

String getPath( );//得到一个文件的路径名

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

String getParent( );/*得到一个文件的上一级目录名*/

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

文件描述

- 文件属性测试

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

boolean canWrite( );//测试当前文件是否可写

boolean canRead( );//测试当前文件是否可读

boolean isFile( );/*测试当前文件是否是文件（不是目录）*/

boolean isDirectory( );/*测试当前文件是否是目录*/

文件描述

- 普通文件信息和工具

  long lastModified( );/*得到文件最近一次修改的时间*/

  long length( );//得到文件的长度，以字节为单位

  boolean delete( );//删除当前文件

- 目录操作

  boolean mkdir( );/*根据当前对象生成一个由该对象指定的路径*/

  String list( );//列出当前目录下的文件


```
import java.io.*;
class FileTest{
    public static void main(String args[]){
        System.out.println("path
    separator"+File.pathSeparator);
        System.out.println("path separator
    char"+File.pathSeparatorChar);
        System.out.println("separator"+File.separator);
        System.out.println("separator char"+File.separatorCha
        File f=new File("/dong1/test1.class");
        System.out.println();
        System.out.println(f);
        System.out.println("exist?"+f.exists());
```


```
System.out.println("name"+f.getName());
System.out.println("path"+f.getPath());
System.out.println("absolute path"+ f.getAbsolutePath());
System.out.println("parent"+f.getParent());
System.out.println("is a file?"+f.isFile());
System.out.println("is a directory?"+f.isDirectory());
System.out.println("length"+f.length());
System.out.println("can read"+f.canRead());
System.out.println("can write"+f.canWrite());
System.out.println("last modified"+f.lastModified());
```

```
        File newF=new File("newFile");
          System.out.println("... Rename "+f+"...");
          f.renameTo(newF);
         System.out.println("name "+newF.getName());
          System.out.println(f+"exist?"+f.exists());
          System.out.println("... delete"+newF+"...");
          newF.delete();
        System.out.println((newF+"exist?"+newF.exists());
    }
}
```

运行结果


```
path separator   :
path separator char:
separator /
separator char /

/dong1/test1.class
exist? true
name test1.class
path /dong1/test1.class
absolute path /dong1/test1.class
parent /dong1
```

运行结果


```
is a file? true
is a directory? false
length 514
can read      true
can write     true
last modified 907117020000
... Rename /dong1/test1.class
name newFile
/dong1/test1.class exist? false
... delete newFile ...
newFile exist? false
```

文件处理

- 列出目录中与某种模式相匹配的文件：

  public String[] list(FilenameFilter filter);

  在接口 FilenameFilter中定义的方法只有：

  boolean accept(File dir,String name);


```
import java.io.*;
public class FileFilterTest{
   public static void main(String args[]){
        File dir=new File("/dongl");
        Filter filter=new Filter("htm");
        System.out.println("list html files in
   directory  "+dir);
       String files[]=dir.list(filter);
```


```
for (int i=0;i<files.length;i++){
   File f=new File(files[i]);
        if(f.isFile())
              System.out.println("file"+f);
          else
              System.out.println("sub directory"+f);
  }
 }
}
```

```
class Filter implements FilenameFilter{
    String extent;
    Filter( String extent){
        this.extent=extent;
    }
    public boolean accept(File dir,String name){
        return name.endsWith(“.”+extent);
    }
}
```

运行结果

list html files in directoty /dongl

file cert_test.htm

file cert_sample.htm

file cert_obj.htm

文件的顺序处理

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

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

随机存取文件

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

随机存取文件

- 构造方法：

  RandomAccessFile(String name,String mode);

  RandomAccessFile(File file,String mode);

- 文件指针的操作

  long getFilePointer( );

  void seek(long pos);

  int skipBytes(int n);

节点流

- 节点流指的是与存储介质直接连接的流，例如FileInputStream,StringReader等。

过滤流

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

过滤流

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

  FilterInputStream(InputStream in);

  FilterOutputStream(OutputStream out);

过滤流

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

```
java.lang.Object
   |
   +----java.io.InputStream
           |
           +----java.io.FilterInputStream
                   |
                   +----java.io.BufferedInputStream
```

过滤流

- DataInputStream 和 DataOutputStream
  可以以与机器无关的格式读取各种类型的数据。
  public class DataInputStream
  extends FilterInputStream
  implements DataInput

过滤流

DataInputStream

byte readByte()

long readLong()

double readDouble()

DataOutputStream

void writeByte(byte)

void writeLong(long)

void writeDouble(double)

过滤流

- LineNumberInputStream

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

- PushbackInputStream

  提供了一个方法可以把刚读过的字节退回到输入流中。

- PrintStream

  打印流的作用是把Java语言的内构类型以其字符表示形式送到相应的输出流。


```
import java.io.*;
public class GetIdentifier{
    public static void main(String args[]){
        try{
          FileInputStream fis=new FileInputStream("test");
          PushbackInputStream pis=new PushbackInputStream(fis);
          int data;
          int idNum=0;
```

```
do{
    data=pis.read();
}while((data!=-1)&&!Character.isLetter((char)data));
pis.unread(data);
while(data=pis.read())!=-1){
    if(Character.isLetterOrDigit((char)data)){
        System.out.print((char)data);
    }else{
                idNum++;
                System.out.println();
```


```
        do{
                data=pis.read();
        }while((data!=-1)&&!Character.isLetter((char)data));
        pis.unread(data);
    }
}
    System.out.println("find"+idNum+ " identifiers");
}catch(FileNotFoundException e){
        System.out.println("error"+e);
```

```
}catch(IOException e){
    System.out.println("error"+e);
    }
}
}
```

文件“test”的内容为:

```
1java is not %only
2#a programming language
```

运行结果

java

is

not

only

a

programming

language

find 7 identifiers

I/O异常

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

  如：FileNotFoundException，EOFException，IOException

流结束的判断

- read()

  返回-1。

- readXXX()

  readInt()、readByte()、readLong()、readDouble()等；

  产生EOFException。

- readLine( )

  返回null。

字符流

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

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

Reader

- Reader类是处理所有字符流输入类的父类。
  - 读取字符

public int read() throws IOException;

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

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

Reader

- 标记流

```
public boolean markSupported();
public void mark(int readAheadLimit) throws
  IOException;
public void reset() throws IOException;
```

- 关闭流

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

Writer

Writer类是处理所有字符流输出类的父类。

– 向输出流写入字符

public void write(int c) throws IOException；

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

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

public void write(String str) throws IOException；

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

Writer

- flush( )

刷空输出流，并输出所有被缓存的字节。

- 关闭流

public abstract void close() throws IOException；

InputStreamReader和OutputStreamWriter

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

InputStreamReader和OutputStreamWriter

- 生成流对象

public InputStreamReader(InputStream in);

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

public OutputStreamWriter(OutputStream out);

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

InputStreamReader和OutputStreamWriter

- 读入和写出字符

  基本同Reader和Writer。
- 获取当前编码方式

  public String getEncoding();
- 关闭流

  public void close() throws IOException;

BufferedReader和BufferedWriter

- 生成流对象

  public BufferedReader(Reader in);

  public BufferedReader(Reader in, int sz);

  public BufferedWriter(Writer out);

  public BufferedWriter(Writer out, int sz);

BufferedReader和BufferedWriter

- 读入/写出字符

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

  public String readLine() throws IOException;

  public void newLine() throws IOException;

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

- 运行结果如下：

  Read: java is a platform independent

  Read: programming language

  Read: it is a

  Read: object oriented language.

从键盘接收输入的数据


```
InputStreamReader ir;
BufferedReader in;
ir=new InputStreamReader(System.in);
in=new BufferedReader(ir);
String s=in.readLine();
System.out.println("Input value is: "+s);
int  i = Integer.parseInt(s);//转换成int型
i*=2;
System.out.println("Input value changed after doubled: "+i);
```

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

ir = new InputStreamReader(is, “8859_1”);

采用ISO 8859_1编码方式，这是一种映射到ASCII码的编码方式，可以在不同平台之间正确转换字符。

对象的串行化(Serialization)

- 对象记录自己的状态以便将来再生的能力，叫作对象的持续性(persistence)。
- 对象通过写出描述自己状态的数值来记录自己，这个过程叫对象的串行化(Serialization)。

串行化方法

- 在java.io包中，接口Serializable用来作为实现对象串行化的工具，只有实现了Serializable的类的对象才可以被串行化。

定义一个可串行化对象


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

构造对象的输入 / 输出流

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

保存对象状态


```
Student stu=new Student(981036,”Liu Ming”,18,
  “CSD”);
FileOutputStream fo
     =new FileOutputStream("data.ser");
ObjectOutputStream so
     =new ObjectOutputStream(fo);
try{
   so.writeObject(stu); so.close();
  }catch(IOExceptione )
         {System.out.println(e);}
```

恢复对象状态


```
FileInputStream fi
    =new FileInputStream("data.ser");
 ObjectInputStream si
    =new ObjectInputStream(fi);
 try{
        stu=(Student)si.readObject();
        si.close();
        }catch(IOExceptione )
        {System.out.println(e);}
```

串行化的注意事项

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

串行化的注意事项

- transient关键字

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

定制串行化

- 缺省的串行化机制，对象串行化首先写入类数据和类字段的信息，然后按照名称的上升排列顺序写入其数值。如果想自己明确地控制这些数值的写入顺序和写入种类，必须定义自己的读取数据流的方式。就是在类的定义中重写writeObject()和readObject()方法。

本讲小结

- 捕获异常
- 声明抛出异常
- 抛出异常
- 字符流
- 字节流
- 对象流
- 节点流
- 过滤流

谢谢！

管道流

- 管道用来把一个程序、线程或代码块的输出连接到另一个程序、线程或代码块的输入
- 管道输入流作为一个通信管道的接收端，管道输出流作为发送端。
- 在使用管道之前，管道输出流和管道输入流必须进行连接。

管道流

- 在构造方法中连接：

  PipedInputStream(PipedOutputStream src);

  PipedOutputStream(PipedInputStream snk);

- 用connect()方法进行连接

  类PipedInputStream中定义为：

  void connetc(PipedOutputStream src);

  类PipedOutputStream中定义为：

  void connetc(PipedInputStream snk);

内存的读写

- ByteArrayInputStream 8位
- ByteArrayOutputStream 8位
- StringBufferInputStream 16位

顺序输入流

- SequenceInputStream 把几个输入流顺序连接起来。


```
import java.io.*;
public class SequenceTest{
    public static void main(String args[]){
        FileInputStream fs1,fs2;
        String s;
        try{
                try{
                fs1=new FileInputStream("text1");
                fs2= new FileInputStream("text2");
```

```
}catch(FileNotFoundExcepton e)
{
System.err.println("File not found or permission denied");
System.exit(-1);
}
SequenceInputStream s1=new SequenceInputStream(fs1,fs2);
DataInputStream s2=new DataInputStream(s1);
while((s=s2.readLine())!=null)
        System.out.println(s);
}catch(IOExceptione)
```


```
{
   System.out.println("error"+e);
   System.exit(-2);
   }
}
}
```