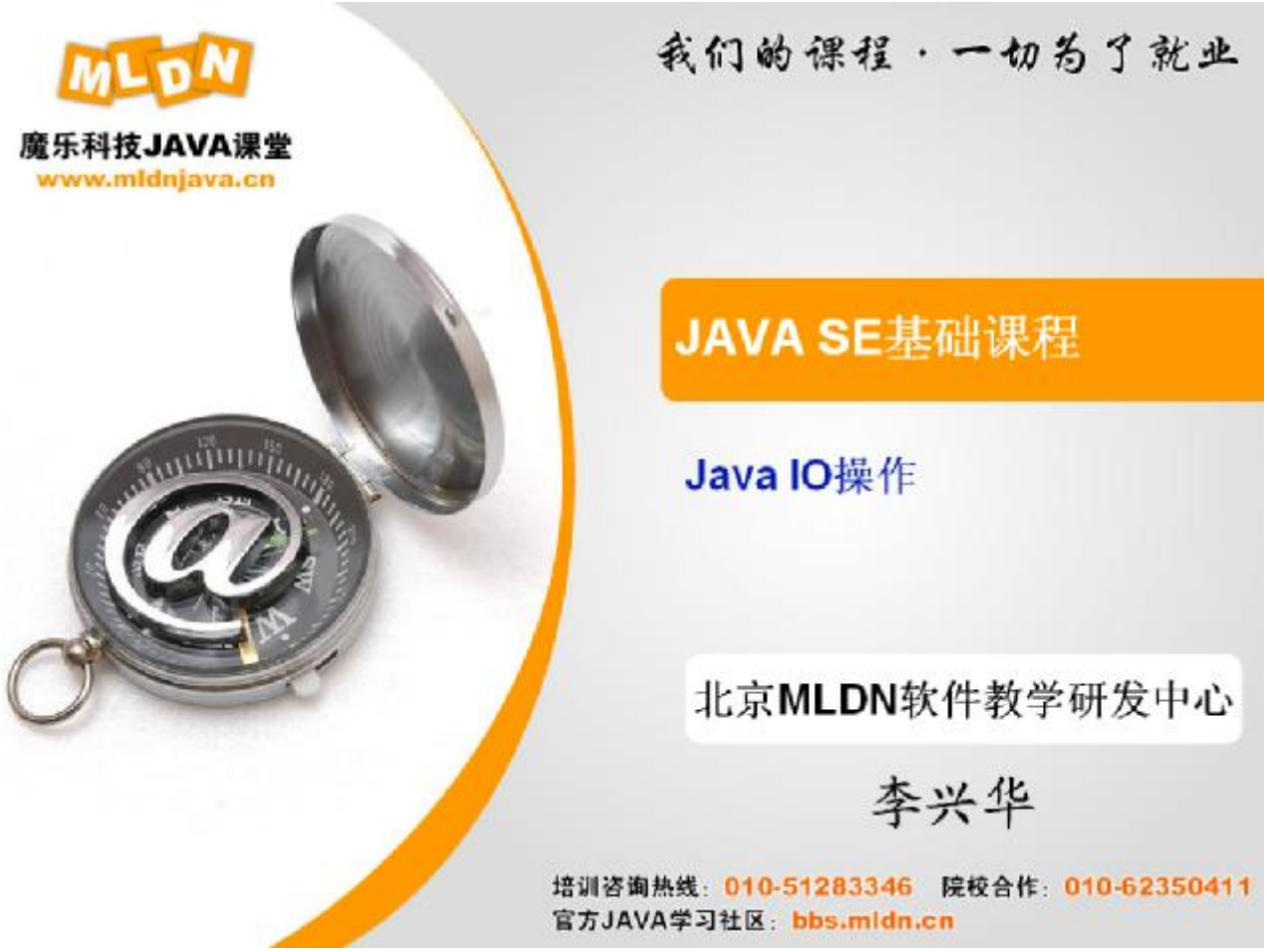
1、课程名称：Java IO 操作



2、知识点

2.1、上次课程的主要知识点

1、 StringBuffer：当字符串需要重复改变的时候可以使用 StringBuffer 类完成，在 String 类中使用“+”完成连接，而 StringBuffer 使用 append()方法完成连接。

2、 垃圾收集：

- 自动回收：JVM 不定期调用 GC 进行垃圾空间的释放
- 手工回收：通过 Runtime 类中的 gc()方法（System.gc()就是调用了 Runtime 类中的 gc()方法）进行回收
- 一个对象回收前会默认调用 finalize()方法，以作为对象的收尾操作。

3、 Date 表示一个日期时间，之后可以通过 SimpleDateFormat()类完成日期的格式化操作，此类可以完成 String 与 Date 型数据的相互转换。

4、 比较器：Comparable、Comparator

- Comparable：在 java.lang 中保存，建立的时候直接实现此接口，有一个 compareTo()方法
- Comparator：在 java.util 中保存，挽救的比较器操作，有两个，比较的是 compare()方法

5、 正则表达式，尤其是 String 类对正则的三个支持。

6、 反射机制：

- Class 实例的三种取得方式
- 通过 Class 类进行对象的实例化操作

7、 大数字：BigInteger、BigDecimal

8、 对象克隆：Cloneable 属于标识接口，一个类需要克隆要覆写 Object 类中的 clone()方法，要扩大权限

2.2、本次预计讲解的知识点

1、 掌握 File 类的使用，并可以使用 File 类进行文件本身的操作；

2、 掌握字节流和字符流的使用，并掌握 IO 操作的基本原理；

3、 掌握打印流、文件操作流、内存操作流的使用

4、 掌握对象序列化的使用。

3、具体内容

IO 操作作为整个 JAVA 中最复杂的开发包，将作为一个难点出现，但是要想跨过此部分，就必须对面向对象的基本概念非常的熟悉，对于抽象类也要熟悉。

根据实例化子类的不同，完成的功能也不同。这句话就是 IO 操作的核心。

整个 IO 包中实际上需要的就是五个类和一个接口：File、OutputStream、InputStream、Writer、Reader；Serializable。所有的类和接口基本上都在 java.io 包中定义的。

3.1、File 类（重点）

File 类在整个 IO 包中是唯一一个与文件本身有关的操作类，所谓的与文件本身有关指的是创建、删除文件等操作。在 java.io 包中的 File 类本身是一个跨平台的文件操作类，所以在操作中要更多的考虑到各个操作系统的区别。

File 类的构造：public File(String pathname)，在建立 File 对象的时候需要指定一个路径。

现在要想创建一个文件，可以使用方法：public boolean createNewFile() throws IOException

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo01 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:\\temp.txt"); // 指定要操作的文件路径
        file.createNewFile();
    }
}
```

既然可以创建文件，那肯定也可以删除：public boolean delete()

但是如果要想删除文件，则肯定要判断文件是否存在：public boolean exists()

那么下面完成这样的一个程序：如果文件存在，则删除掉，如果文件不存在，则创建新的。

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo01 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:\\temp.txt"); // 指定要操作的文件路径
        if (file.exists()) {
            file.delete(); // 删除文件
        } else {
            file.createNewFile(); // 创建新文件
        }
    }
}
```

但是在创建和删除文件的时候发现会出现延迟的问题，因为 JAVA 运行机制是运行在 JVM 上，由 JVM 进行 OS 的具体的适应，所以中间存在延迟，而且本程序也有问题，在 Java 的最大特点是可移植性，但是在不同的操作系统中路径的分割符肯定是不一样的：

- windows 中使用“\”
- linux 中使用“/”

那么要想解决这样的问题，就必须观察 File 类定义的常量：public static final String separator。

separator 是一个常量，按照常量的命名要求肯定全部的字母都要大写：SEPARATOR。这些都是由于 Java 发展的历史原因所造成的问题。

```
File file = new File("d:" + File.separator + "temp.txt"); // 指定要操作的文件路径
```

由于在给定文件的路径上有可能给出的是一个文件，也有可能给出的是一个文件夹，那么为了判断，在 File 类中提供了以下的两个方法：

- 判断是否是文件： public boolean isFile()
- 判断是否是文件夹： public boolean isDirectory()

范例：判断类型

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo02 {
    public static void main(String[] args) throws Exception {
        File file1 = new File("d:" + File.separator + "temp.txt"); // 指定要操作的文件路径
        File file2 = new File("d:" + File.separator + "testjava"); // 指定要操作的文件路径
        System.out.println(file1.isFile());
        System.out.println(file2.isDirectory());
    }
}
```

在 File 类的操作中可以通过代码列出一个文件夹之中的完整内容，方法如下：

- 列出内容：public **String[]** list()
- 列出内容：public **File[]** listFiles()

范例：使用 list()方法

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo03 {
    public static void main(String[] args) throws Exception {
```

```
        File file = new File("d:" + File.separator + "testjava"); // 指定要操作的文件路径
        if (file.isDirectory()) { // 如果是文件夹，则列出内容
            String list[] = file.list(); // 列出全部的内容
            for (int x = 0; x < list.length; x++) {
                System.out.println(list[x]);
            }
        }
    }
}
```

现在显示出来的只是文件或文件夹的名称。

范例：使用 listFiles()完成列表

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo03 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "testjava"); // 指定要操作的文件路径
        if (file.isDirectory()) { // 如果是文件夹，则列出内容
            File list[] = file.listFiles() ;   // 列出全部内容
            for (int x = 0; x < list.length; x++) {
                System.out.println(list[x]);
            }
        }
    }
}
```

发现使用 listFiles()方法输出的时候可以输出一个完整的路径，而且返回的是 File 类的对象，可以进行更多的操作。

```
package org.lxh.filedemo;
import java.io.File;
public class FileDemo03 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "testjava"); // 指定要操作的文件路径
        if (file.isDirectory()) { // 如果是文件夹，则列出内容
            File list[] = file.listFiles() ;   // 列出全部内容
            for (int x = 0; x < list.length; x++) {
                System.out.println(list[x] + " --> " + list[x].length());
            }
        }
    }
}
```

思考题：

现在要求输出一个给定目录中的全部文件的路径。

本程序肯定只能依靠递归的操作完成，因为在一个给定的路径下有可能还是文件夹，那么如果是文件夹的话则肯定要继续列出，重复判断。

```
package org.lxh.filedemo;
import java.io.File;
```

```
public class ListFileDemo {
    public static void main(String[] args) {
        File file = new File("D:" + File.separator);
        list(file);
    }
    public static void list(File file) {
        if (file.isDirectory()) { // 如果是文件夹，则继续列出
            File f[] = file.listFiles();
            if (f != null) { // 文件夹的内容已经列出
                for (int x = 0; x < f.length; x++) {
                    list(f[x]); // 继续列出每一个内容
                }
            }
        }
        System.out.println(file);
    }
}
```

包括 windows 的资源管理器就是采用了这种形式的代码完成的。

3.2、字节流与字符流（核心重点）

File 类本身是与文件操作有关，但是如果要想操作内容则必须使用字节流或字符流完成，但是不管是使用何种的输入输出流，其基本的操作原理是一样的（以文件流为准）

1、 使用 File 类找到一个文件
2、 通过字节流或字符流的子类进行对象的实例化
3、 进行读或写的操作
4、 关闭字节或字符流

由于流的操作属于资源操作，所以在操作的最后一定要关闭以释放资源。

操作流有以下几个：

- 字节流：OutputStream、InputStream
- 字符流：Writer、Reader

3.2.1、字节输出流

字节输出流使用的是 OutputStream，此类定义如下：

```
public abstract class OutputStream extends Object implements Closeable, Flushable
```

本类是一个抽象类，根据面向对象的概念，要通过子类进行对象的实例化操作。

在此类中定义了如下的几个常用方法：

- 关闭流：public void close() throws IOException
- 写一组数据：public void write(byte[] b) throws IOException
- 写一个数据：public void write(int b) throws IOException

但是要想为 OutputStream 实例化，且进行文件操作的话，就要使用 FileOutputStream 子类。

- 构造：public FileOutputStream(**File file**) throws FileNotFoundException

范例：使用字节流进行输出，输出“Hello World”。

```
package org.lxh.outputstreamdemo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
public class OutputStreamDemo01 {
    public static void main(String[] args) throws Exception {
        // 1、通过File找到一个文件
        File file = new File("d:" + File.separator + "temp.txt");
        // 2、实例化OutputStream对象
        OutputStream out = new FileOutputStream(file);
        String info = "Hello World!!!"; // 要输出的字符串
        byte data[] = info.getBytes(); // 将字符串变为字节数组
        out.write(data); // 输出内容
        out.close();
    }
}
```

现在已经可以向文件中输出内容了，但是此时程序每执行一次，实际上都会输出，但是属于覆盖的操作，如果要想在文件的尾部追加的话，则必须观察 FileOutputStream 类的另外一个构造：

- 追加：public FileOutputStream(File file,boolean append) throws FileNotFoundException

如果需要在追加的上面加入换行的话，使用“\r\n”。

```
package org.lxh.outputstreamdemo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
public class OutputStreamDemo01 {
    public static void main(String[] args) throws Exception {
        // 1、通过File找到一个文件
        File file = new File("d:" + File.separator + "temp.txt");
        // 2、实例化OutputStream对象
        OutputStream out = new FileOutputStream(file, true); // 追加
        String info = "\r\nHello World!!!"; // 要输出的字符串
        byte data[] = info.getBytes(); // 将字符串变为字节数组
        out.write(data); // 输出内容
        out.close();
    }
}
```

3.2.2、字节输入流

程序中可以使用 OutputStream 进行输出的操作，也可以使用 InputStream 完成输入的操作，此类定义如下：

```
public abstract class InputStream extends Object implements Closeable
```

此类依然是一个抽象类，肯定要使用子类完成，如果是文件输入，使用 FileInputStream 类。

InputStream 类中定义的方法如下：

- 关闭：public void close() throws IOException
- 读取一个字节：public abstract int read() throws IOException
- 读取一组内容：public int read(byte[] b) throws IOException

如果要读，则肯定需要一个数组，数组肯定要首先开辟好大小，用于接收内容。

但是，与 OutputStream 类似，要读取就要观察 FileInputStream 类的构造方法：

- 构造：public FileInputStream(File file) throws FileNotFoundException

```
package org.lxh.inputstreamdemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class InputStreamDemo01 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "temp.txt");
        InputStream input = new FileInputStream(file);
        byte data[] = new byte[1024]; // 开辟一个空间
        int len = input.read(data); // 接收输入流的内容
        System.out.println("内容是：（" + new String(data, 0, len) + "）");
    }
}
```

以上的代码属于一次性全部读取，但是在 InputStream 类中也可以每次读取一个字节。

```
package org.lxh.inputstreamdemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class InputStreamDemo02 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "temp.txt");
        InputStream input = new FileInputStream(file);
        byte data[] = new byte[1024]; // 开辟一个空间
        int len = 0; // 记录读取的长度
        int temp = 0;
        do {
            temp = input.read(); // 读取一个字节
            if (temp != -1) { // 如果不为-1表示内容可以增加
                data[len++] = (byte) temp; // 保存在字节数组中
            }
        } while (temp != -1); // 如果不是-1表示还有内容可以读
        System.out.println("内容是：（" + new String(data, 0, len) + "）");
    }
}
```

但是以上的读取方式在开发中会变成另外一种代码形式：

```
        int temp = 0;
        while ((temp = input.read()) != -1) {
```

```
            data[len++] = (byte) temp;
        }
```

以上的程序的语句形式为以后使用的重点。

3.2.3、字符输出流

Writer 属于字符输出流，Writer 类也是一个抽象类，既然要操作文件，肯定使用 FileWriter。

```
package org.lxh.writerdemo;
import java.io.File;
import java.io.FileWriter;
import java.io.Writer;
public class WriterDemo01 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "temp.txt");
        Writer out = new FileWriter(file);
        out.write("Hello World"); // 直接输出字符串
        out.close() ;
    }
}
```

3.2.4、字符输入流

Reader 也肯定是一个抽象类，要输入文件使用 FileReader。

```
package org.lxh.readerdemo;
import java.io.File;
import java.io.FileReader;
import java.io.Reader;
public class ReaderDemo01 {
    public static void main(String[] args) throws Exception {
        File file = new File("d:" + File.separator + "temp.txt");
        Reader read = new FileReader(file);
        char data[] = new char[1024];
        int len = read.read(data);
        System.out.println(new String(data, 0, len));
    }
}
```

3.2.5、字节流和字符流的区别

字节流和字符流在使用上的代码结构都是非常类似的，但是其内部本身也是有区别的，因为在进行字符流操作的时候会使用到缓冲区，而字节流操作的时候是不会使用到缓冲区的。

在输出的时候，OutputStream 类即使最后没有关闭内容也可以输出。但是如果是 Writer 的话，则如果不关闭，最后一

条内容是无法输出的，因为所有的内容都是保存在了缓冲区之中，每当调用了 close()方法就意味着清空缓冲区了。那么可以证明字符流确实使用了缓冲区：

- 字节流：程序 à 文件
- 字符流：程序 à 缓冲区 à 文件

如果现在字符流即使不关闭也可以完成输出的话，则必须强制性清空缓冲区：

- 方法：public void flush() throws IOException

两者相比，肯定使用字节流更加的方便，而且在程序中像图片、MP3 等都是采用字节的方式的保存，那么肯定字节流会比字符流使用的更广泛。

但是需要说明的是，如果要是想操作中文的话，字符流肯定是最好使的。

3.2.6、思考题

现在要求完成一个 Copy 程序，完全模仿 DOS 中的拷贝命令。

命令的运行形式，可以通过初始化参数的方式设置两个路径

- 形式：java Copy 源文件路径 目标文件路径

本程序完成有两个思路：

- 思路一：将所有的内容全部读取进来，之后一次性保存
- 思路二：边读边写

而且要进行读取的时候还要判断源文件是否存在。

```
package org.lxh.copydemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
public class Copy {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.out.println("语法命令不正确！");
            System.exit(1);
        }
        File file1 = new File(args[0]);
        if (file1.exists()) { // 如果源文件存在
            File file2 = new File(args[1]);
            InputStream input = new FileInputStream(file1);
            OutputStream output = new FileOutputStream(file2);
            int temp = 0;
            while ((temp = input.read()) != -1) { // 边读边写
                output.write(temp); // 输出
            }
            input.close();
            output.close();
            System.out.println("文件拷贝完成。");
```

```
        }
    }
}
```

一定要明白的是，OutputStream、InputStream 把握住之后，IO 的核心就把握了。

3.3、内存操作流（重点）

之前的文件操作流是以文件的输入输出为主的，但是如果现在将输入输出的位置一改变，改变成了内存，那么就称为内存操作流。使用 ByteArrayInputStream 完成内存的输入操作，而使用 ByteArrayOutputStream 完成内存的输出操作，但是一定要注意的是，现在的输入和输出都是以内存为标准的。

ByteArrayInputStream 构造：public ByteArrayInputStream(byte[] buf)

```
package org.lxh.bytearraydemo;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
public class ByteArrayDemo {
    public static void main(String[] args) throws Exception {
        String info = "helloworld";
        InputStream input = new ByteArrayInputStream(info.getBytes());
        OutputStream output = new ByteArrayOutputStream();
        int temp = 0;
        while ((temp = input.read()) != -1) {
            output.write(Character.toUpperCase((char) temp));
        }
        String str = output.toString(); // 取出内容
        input.close() ;
        output.close() ;
        System.out.println(str) ;
    }
}
```

现在虽然完成了内存的操作，但是可以发现现在的 IO 都是从内存中，也就是说可以将内存当作一个临时的文件进行操作，所以内存操作流一般在产生临时文件内容的时候使用。

以后在学习 AJAX + XML 操作的时候就会使用到内存操作流。

3.4、打印流（重点）

思考：如果现在要想完成一个字符串或者是 boolean 型或者是字符型的数据输出使用 OutputStream 是否方便？

肯定是不方便的，因为 OutputStream 中只能操作字节数据，所以其他的数据类型很难操作，那么在 Java 的 IO 包中为了解决这种问题增加了两种类：PrintStream、PrintWriter。

观察 PrintStream 类的构造：public PrintStream(OutputStream out)

虽然 PrintStream 是 OutputStream 的子类，但是在实例化的时候依然需要一个 OutputStream 的对象。

在 PrintStream 中定义了一系列的输出操作，可以方便的完成输出，那么这种设计思路就是装饰设计。

```
package org.lxh.printstreamdemo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
public class PrintStreamDemo01 {
    public static void main(String[] args) throws Exception {
        PrintStream out = new PrintStream(new FileOutputStream(new File("d:"
                + File.separator + "test.txt")));
        out.print(1 + " + " + 1 + " = ");
        out.println(1 + 1);
        out.println("Hello World!!!") ;
    }
}
```

在开发中由于 PrintStream 较为好用，所以只要是输出就使用打印流完成。

但是在 JDK 1.5 之后打印流也被增强了，C 语言中的打印方法是 printf()，在 PrintStream 中也具备了。

一些数据的表示格式：%d、%f、%s

```
package org.lxh.printstreamdemo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.PrintStream;
public class PrintStreamDemo02 {
    public static void main(String[] args) throws Exception {
        PrintStream out = new PrintStream(new FileOutputStream(new File("d:"
                + File.separator + "test.txt")));
        String name = "张三";
        int age = 30;
        float score = 90.89765f;
        out.printf("姓名：%s，年龄：%d，成绩：%5.2f", name, age, score);
    }
}
```

而且再提醒的是，在 JDK 1.5 之后的 String 类也有变化，也增加了同样的方法：

```
public static String format(String format,Object... args)
```

3.5、System 类对 IO 的支持（重点）

在 System 类中定义了以下的三个常量：

- 错误输出：public static final **PrintStream** err
- 屏幕输出：public static final **PrintStream** out
- 键盘输入：public static final InputStream in

3.5.1、错误输出

System.err 表示的是错误的输出，此类型也属于 PrintStream 类型。

```
package org.lxh.systemdemo;
public class SystemErrDemo {
    public static void main(String[] args) {
        try {
            Integer.parseInt("A");
        } catch (NumberFormatException e) {
            System.err.println(e);
        }
    }
}
```

使用 System.err 和 System.out 的输出是完全一样的，但是在 Eclipse 中 System.err 的输出会使用红色表示。

两个的区别只是人为的增加的区别，System.err 一般是不希望用户看见的错误，而 System.out 是希望用户看见的错误。

3.5.2、屏幕输出

System.out 属于屏幕输出的操作，对应着显示器，但是可以通过 System.out 为 OutputStream 实例化。

```
package org.lxh.systemdemo;
import java.io.OutputStream;
public class SystemOutDemo {
    public static void main(String[] args) throws Exception {
        OutputStream out = System.out ;
        out.write("hello world".getBytes()) ;
    }
}
```

3.5.3、键盘输入

一般的语言中都存在着键盘输入数据的操作，在 Java 中也有，只是一般来讲这种操作不是现成的，而且必须依靠 IO 流的操作才能完成，输入靠的是 System.in。

```
package org.lxh.systemdemo;
import java.io.InputStream;
public class SystemInDemo {
    public static void main(String[] args) throws Exception {
        InputStream input = System.in; // 可以由键盘输入
        byte data[] = new byte[1024];
        System.out.print("请输入内容：");
        int len = input.read(data);
        System.out.println("输入的内容是：" + new String(data, 0, len));
    }
}
```

此时已经完成了内容的输入，但是本程序有问题，因为现在所有的输入内容都是保存在了 data 这个字节数组之中，如果现在输入的内容的长度大于数组中的长度的话，则超出的部分无法接收。

这种操作出现问题的主要原因是一开始的数组大小限制了长度，那么如果现在没有长度限制呢？

```
package org.lxh.systemdemo;
import java.io.InputStream;
public class SystemInDemo {
    public static void main(String[] args) throws Exception {
        InputStream input = System.in; // 可以由键盘输入
        StringBuffer buf = new StringBuffer();
        System.out.print("请输入内容：");
        int temp = 0;
        while ((temp = input.read()) != -1) {
            char c = (char) temp;
            if (c == '\n') {
                break ;
            }
            buf.append(c);
        }
        System.out.println("输入的内容是：" + buf);
    }
}
```

这种输入现在只能适合于英文环境上，因为是按照每一个字节的方式读取的，而一个中文是两个字节。

所以，如果要想彻底的解决问题，最好的做法是，将所有的输入内容暂时存放到一个缓冲区之中，之后从缓冲区里一次性将内容全部读取回来。

3.6、BufferedReader（理解）

BufferedReader 的主要功能是用于缓冲区读取的，但是有一点，从类的定义上可以发现，此类属于字符输入流，而 System.in 属于字节流，那么很明显，如果要想使用 BufferedReader 就需要将一个字节流变成字符流，为了解决这样的问题，在 Java 中提供了以下的两个转换类：

- 将输入的字节流变为字符流：InputStreamReader
- 将输出的字符流变为字节流：OutputStreamWriter

在 BufferedReader 类中提供了一个专门的读取操作：public String readLine() throws IOException

```
package org.lxh.bufferdemo;
import java.io.BufferedReader;
import java.io.InputStreamReader;
public class BufferDemo {
    public static void main(String[] args) throws Exception {
        BufferedReader buf = new BufferedReader(
                new InputStreamReader(System.in));
        System.out.print("请输入内容：");
        String info = null; // 用于接收输入的内容
        info = buf.readLine(); // 接收内容
        System.out.println(info);
    }
}
```

由于所有的内容都是从缓冲区一次性读取完成的，那么不会出现乱码问题。

以上的代码就是一个键盘输入的标准程序。

3.7、思考题

现在要求从键盘输入两个数字，之后分别进行四则运算。

如果用户输入的内容有错误，则应该提示用户后让用户重新输入。

本程序要求考虑类的设计结构问题。

本程序应该设计如下几个类：

- MyMath：执行具体的四则运算
- InputData：输入数据类型
- Operate：接收输入数据，执行计算
- Main：主类

3.8、Scanner 类

在 JDK 1.5 之后为了方便输入，又增加了一个新的类：Scanner，此类并不是在 java.io 包中定义的，而是在 java.util 包中定义的。

使用 Scanner 可以方便的完成各种字节输入流的输入，构造：public Scanner(InputStream source)

```
package org.lxh.scandemo;
import java.util.Scanner;
public class ScannerDemo01 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("请输入内容：");
        if (scan.hasNext()) { // 现在有内容
            String str = scan.next();
            System.out.println("输入的内容是：" + str);
        }
    }
}
```

除了这种功能之外，使用 Scanner 也可以输入各种数据类型，例如：float。

```
package org.lxh.scandemo;
import java.util.Scanner;
public class ScannerDemo02 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("请输入内容：");
        if (scan.hasNextInt()) { // 现在有内容
            int str = scan.nextInt();
            System.out.println("输入的内容是：" + str);
        }
    }
}
```

但是，如果要想进行其他数据格式验证的话，则就需要编写正则表达式完成了。

```
package org.lxh.scandemo;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
public class ScannerDemo03 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("请输入日期：");
        if (scan.hasNext("\\d{4}-\\d{2}-\\d{2}")) { // 现在有内容
            String str = scan.next("\\d{4}-\\d{2}-\\d{2}");
            Date date = null;
            try {
                date = new SimpleDateFormat("yyyy-MM-dd").parse(str);
            } catch (ParseException e) {
                e.printStackTrace();
            }
            System.out.println(date);
        }
    }
}
```

使用 Scanner 类除了可以完成以上的输入之外，也可以从一个文件流中输入内容。

```
package org.lxh.scandemo;
import java.io.File;
import java.io.FileInputStream;
import java.util.Scanner;
public class ScannerFileDemo {
    public static void main(String[] args) throws Exception {
        Scanner scan = new Scanner(new FileInputStream(new File("d:"
                + File.separator + "test.txt")));
        scan.useDelimiter("\n"); // 将换行作为分隔符
        while (scan.hasNext()) {
            String str = scan.next();
            System.out.print(str);
        }
    }
}
```

在 IO 操作记住：输出的时候使用 PrintStream，输入的时候使用 Scanner。

3.9、对象序列化（核心重点）

对象序列化的是一个在 JAVA EE 开发中使用最多的概念，而且在以后的分布式开发中使用较多。

3.9.1 对象序列化的概念（重点）

所谓的对象序列化就是指将一个在内存中保存的对象变成一个二进制的数据流进行传输。但并不是所有类的对象都可以进行序列化的操作，如果一个对象需要被序列化，则对象所在的类必须实现 **Serializable** 接口。但是此接口中没有任何的方法定义，所以此接口和 Cloneable 接口是完全一样的，都是作为标识接口出现。

```
package org.lxh.serdemo;
import java.io.Serializable;
@SuppressWarnings("serial")
public class Person implements Serializable {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public String toString() {
        return "姓名：" + this.name + "，年龄：" + this.age;
    }
}
```

此时，Person 类的对象已经允许被序列化操作了，即变为二进制的 byte 流。

如果要想序列化，需要使用 ObjectOutputStream 类完成。如果需要反序列化，则需要 ObjectInputStream 类完成。

3.9.2、序列化对象

ObjectOutputStream 主要是序列化对象的使用，也是一个 OutputStream 的子类，方法：

- 构造：public ObjectOutputStream(OutputStream out) throws IOException
- 输出对象：public final void writeObject(Object obj) throws IOException

```
package org.lxh.serdemo;
import java.io.File;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class ObjectOutputStreamDemo {
    public static void main(String[] args) throws Exception {
        Person per = new Person("张三", 30);
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(
                new File("D:" + File.separator + "person.ini")));
        oos.writeObject(per); // 序列化对象
        oos.close();
    }
}
```

3.9.3、反序列化对象

ObjectInputStream 就可以完成对象的反序列化的功能，方法如下：

- 构造：public ObjectInputStream(InputStream in) throws IOException
- 读取对象：public final Object readObject() throws IOException,ClassNotFoundException

```
package org.lxh.serdemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class ObjectInputStreamDemo {
    public static void main(String[] args) throws Exception {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
                new File("D:" + File.separator + "person.ini")));
        Object obj = ois.readObject();
        ois.close();
        if (obj instanceof Person) {
            Person per = (Person) obj;
            System.out.println(per);
        }
    }
}
```

3.9.4、transient 关键字（重点）

实际上在进行对象序列化的时候，序列化的是类中的属性，因为每个类的对象只有属性是不同的。

但是如果现在有某个属性不希望被序列化下来的话，则可以使用 transient 关键字。

```
    private transient String name;
    private int age;
```

3.9.5、序列化一组对象

在正常情况下 ObjectOutputStream 只能输出一个对象，如果现在有多个对象要同时进行序列化的话，则可以使用对象数组的形式完成，对象数组可以直接使用 Object 接收。

```
package org.lxh.serdemo;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
public class ObjectArrayDemo {
    public static void main(String[] args) throws Exception {
        ser(new Person[] { new Person("张三", 20), new Person("李四", 21),
```

```
                new Person("王五", 22) });
        Person per[] = (Person[]) dser();
        for (int x = 0; x < per.length; x++) {
            System.out.println(per[x]);
        }
    }
    public static void ser(Object obj) throws Exception {
        ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(
                new File("D:" + File.separator + "person.ini")));
        oos.writeObject(obj);
        oos.close();
    }
    public static Object dser() throws Exception {
        ObjectInputStream ois = new ObjectInputStream(new FileInputStream(
                new File("D:" + File.separator + "person.ini")));
        Object obj = ois.readObject();
        ois.close();
        return obj;
    }
}
```

此时也可以发现对象数组本身是存在长度的限制，所以可以通过链表继续完成优化。

4、总结

1、 使用 File 类可以操作文件，但是在编写文件分隔符的时候要注意使用 File.separator
2、 字节流和字符流最大的区别在于字符流使用到了缓冲区操作，但是一般字节流使用较多
3、 内存流是所有的输入输出都在内存中完成：ByteArrayInputStream、ByteArrayOutputStream
4、 如果要输出内容可以使用 PrintStream 类方便的完成
5、 如果要输入内容可以使用 Scanner 类
6、 对象序列化

5、预习任务

下次讲解 Java 类集框架，要求预习的内容：Collection、List、Set、Map、ArrayList、Vector、HashSet、TreeSet、Hashtable、HashMap、Iterator、ListIterator、Enumeration、Properties。
复习好 Comparable 接口的使用、泛型的操作

6、作业

留一道简单程序

A、功能描述：

有一个班采用民主投票方法推选班长，班长候选人共 4 位，每个人姓名及代号分别为 张三 1，李四 2，王五 3，刘六 4。程序操作员将每张选票上所填的代号（1、2、3、或 4）循环输入电脑，输入数字 0 结束输入，然后将所有候选人的得票情况显示出来，并显示最终当选者的信息。

B、具体要求如下：

（1）、要求用面向对象方法，编写候选人类 Candidate，将候选人姓名、代号和票数保存到类 Candidate(候选人类)中，并实现相应的 getXXX 和 setXXX 方法。

（2）、编写主程序 class OneTest

（3）、输入数据之前，显示出各位候选人的代号及姓名：（提示：建立一个候选人类型数组）如下图所示。

（4）、循环执行接收键盘输入的班长候选人代号，直到输入的数字为 0，结束选票的输入工作，如下图所示

（5）、在接收每次输入的选票后要求验证该选票是否有效，即：如果输入的数不是 0，1，2，3，4 这 5 个数字之一，或者输入一串字母（捕捉异常），应显示出错误提示信息：此选票无效，请输入正确的候选人代号！并继续等待输入。

（6）、输入结束后显示所有候选人的得票情况，如图所示

（7）、输出最终当选者的相关信息，如图所示。

C、参考图示：

```
1：张三【0票】
2：李四【0票】
3：王五【0票】
4：刘六【0票】
请输入班长候选人代号（数字0结束）：1
请输入班长候选人代号（数字0结束）：1
请输入班长候选人代号（数字0结束）：1
请输入班长候选人代号（数字0结束）：2
请输入班长候选人代号（数字0结束）：3
请输入班长候选人代号（数字0结束）：4
请输入班长候选人代号（数字0结束）：5
此选票无效，请输入正确的候选人代号！
请输入班长候选人代号（数字0结束）：hello
此选票无效，请输入正确的候选人代号！
请输入班长候选人代号（数字0结束）：0
1：张三【4票】
2：李四【1票】
3：王五【1票】
4：刘六【1票】
投票最终结果：张三同学，最后以4票当选班长！
```