Потоки ввода/вывода

Программы, написанные нами в предыдущих лабораторных работах, воспринимали информацию только из параметров командной строки и графических компонентов, а результаты выводили на консоль или в графические компоненты. Однако во многих случаях требуется выводить результаты на принтер, в файл, базу данных или передавать по сети. Исходные данные тоже часто приходится загружать из файла, базы данных или из сети.

Для того чтобы отвлечься от особенностей конкретных устройств ввода/вывода, в Java употребляется понятие *потока* (stream). Считается, что в программу идет *входной поток* (input stream) символов Unicode или просто байтов, воспринимаемый в программе методами read(). Из программы методами write о или print (), println() выводится *выходной поток* (output stream) символов или байтов. При этом неважно, куда направлен поток: на консоль, на принтер, в файл или в сеть, методы write () и print () ничего об этом не знают.

Можно представить себе поток как трубу, по которой в одном направлении последовательно "текут" символы или байты, один за другим. Методы read () , write () , print (), println () взаимодействуют с одним концом трубы, другой конец соединяется с источником или приемником данных конструкторами классов, в которых реализованы эти методы.

Конечно, полное игнорирование особенностей устройств ввода/вывода сильно замедляет передачу информации. Поэтому в Java все-таки выделяется файловый ввод/вывод, вывод на печать, сетевой поток.

Три потока определены в классе system статическими полями in, out и err. Их можно использовать без всяких дополнительных определений, что мы и делали на протяжении всей книги. Они называются соответственно *стандартным вводом* (stdin), *стандартным выводом* (stdout) и *стандартным выводом сообщений* (stderr). Эти стандартные потоки могут быть соединены с разными конкретными устройствами ввода и вывода.

Потоки out и err — это экземпляры класса Printstream, организующего выходной поток байтов. Эти экземпляры выводят информацию на консоль методами print (), println () и write (), которых в классе Printstream имеется около двадцати для разных типов аргументов.

Поток err предназначен для вывода системных сообщений программы: трассировки, сообщений об ошибках или, просто, о выполнении каких-то этапов программы. Такие сведения обычно заносятся в специальные журналы, log-файлы, а не выводятся на консоль. В Java есть средства переназначения потока, например, с консоли в файл.

Поток in — это экземпляр класса inputstream. Он назначен на клавиатурный ввод с консоли методами read(). Класс inputstream абстрактный, поэтому реально используется какой-то из его подклассов.

Понятие потока оказалось настолько удобным и облегчающим программирование ввода/вывода, что в Java предусмотрена возможность создания потоков, направляющих символы или байты не на внешнее устройство, а в массив или из массива, т. е. связывающих программу с областью оперативной памяти. Более того, можно создать поток, связанный со строкой типа string, находящейся, опять-таки, в оперативной памяти. Кроме того, можно создать *канал* (pipe) обмена информацией между подпроцессами.

Еще один вид потока — поток байтов, составляющих объект Java. Его можно направить в файл или передать по сети,'а потом восстановить в оперативной памяти. Эта операция называется *сериализацией* (serialization) объектов.

Методы организации потоков собраны в классы пакета java.io.

Кроме классов, организующих поток, в пакет java.io входят классы с методами преобразования потока, например, можно преобразовать поток байтов, образующих целые числа, в поток этих чисел.

Еще одна возможность, предоставляемая классами пакета java.io, — слить несколько потоков в один поток.

Итак, в Java есть целых четыре иерархии классов для создания, преобразования и слияния потоков. Во главе иерархии четыре класса, непосредственно расширяющих класс object:

- Reader — абстрактный класс, в котором собраны самые общие методы символьного ввода;
- writer — абстрактный класс, в котором собраны самые общие методы символьного вывода;
- inputstream — абстрактный класс с общими методами байтового ввода;
- Outputstream — абстрактный класс с общими методами байтового вывода.

Классы входных потоков Reader и inputstream определяют по три метода ввода:

- read () — возвращает один символ или байт, взятый из входного потока, в виде целого значения типа int; если поток уже закончился, возвращает -1;
- read (chart] buf) — заполняет заранее определенный массив buf символами из входного потока; в классе inputstream массив типа bytet] и заполняется он байтами; метод возвращает фактическое число взятых из потока элементов или -1, если поток уже закончился;
- read (char[] buf, int offset, int len) — заполняет часть символьного или байтового массива buf, начиная с индекса offset, число взятых из потока элементов равно len; метод возвращает фактическое число взятых из потока элементов или -1.

Эти методы выбрасывают IOException, если произошла ошибка ввода/вывода.

Четвертый метод skip (long n) "проматывает" поток с текущей позиции на п символов или байтов вперед. Эти элементы потока не вводятся методами read(). Метод возвращает реальное число пропущенных элементов, которое может отличаться от п, например поток может закончиться.

Текущий элемент потока можно пометить методом mark (int n), а затем вернуться к помеченному элементу методом reset о, но не более чем через п элементов. Не все подклассы реализуют эти методы, поэтому перед расстановкой пометок следует обратиться к логическому методу marksupported (), который возвращает true, если реализованы методы расстановки и возврата к пометкам.

Классы выходных потоков writer и outputstream определяют по три почти одинаковых метода вывода:

- write (char[] buf) — выводит массив в выходной поток, в классе Outputstream массив имеет тип byte[];
- write (char[] buf, int offset, int len) — выводит len элементов массива buf, начиная с элемента с индексом offset;
- write (int elem) в классе Writer - выводит 16, а в классе Outputstream 8 младших битов аргумента elem в выходной поток,

В классе writer есть еще два метода:

- write (string s) — выводит строку s в выходной поток;
- write (String s, int offset, int len) — выводит len символов строки s, начиная с символа с номером offset.

Многие подклассы классов writer и outputstream осуществляют буферизованный вывод. При этом элементы сначала накапливаются в буфере, в оперативной памяти, и выводятся в выходной поток только после того, как буфер заполнится. Это удобно для выравнивания скоростей вывода из программы и вывода потока, но часто надо вывести информацию в поток еще до заполнения буфера. Для этого предусмотрен метод flush о. Данный метод сразу же выводит все содержимое буфера в поток.

Наконец, по окончании работы с потоком его необходимо закрыть методом closed.

Классы, входящие в иерархии потоков ввода/вывода, показаны на рис. 1 и. 2.

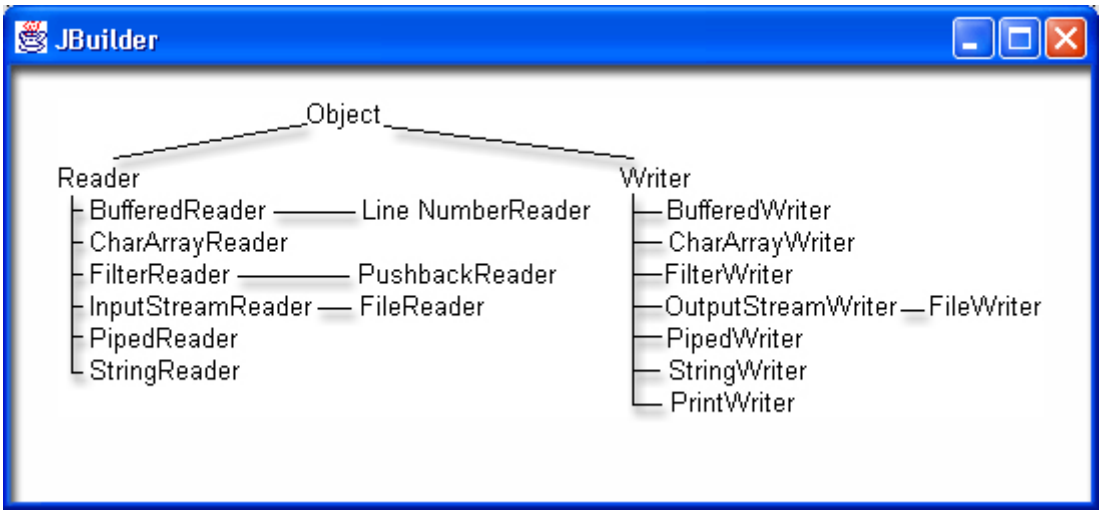

Рис. 18.1. Иерархия символьных потоков

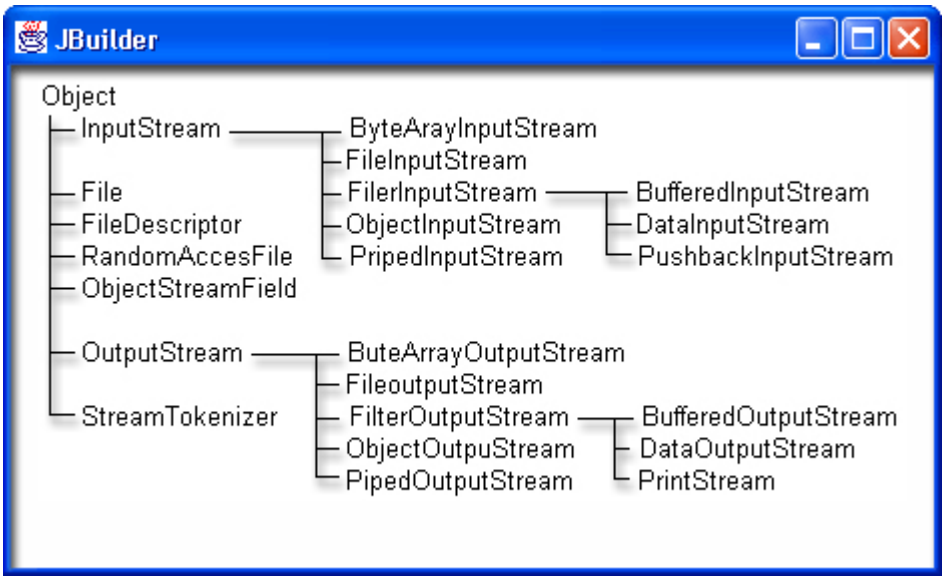

Рис. 2. Классы байтовых потоков

Все классы пакета java.io можно разделить на две группы: классы, создающие поток (data sink), и классы, управляющие потоком (data processing).

Классы, создающие потоки, в свою очередь, можно разделить на пять групп:

- классы, создающие потоки, связанные с файлами:

  FileReader FileInputStream

  FileWriterFile Outputstream

  RandomAccessFile

- классы, создающие потоки, связанные с массивами:

CharArrayReader ByteArraylnputStream

CharArrayWriter ByteArrayOutputStream

- классы, создающие каналы обмена информацией между подпроцессами:

  PipedReader PipedlnputStream

  PipedWriter PipedOutputStream

- классы, создающие символьные потоки, связанные со строкой:

  StringReader

  StringWriter

- классы, создающие байтовые потоки из объектов Java:

  ObjectlnputStream

  ObjectOutputStream

Слева перечислены классы символьных потоков, справа — классы байтовых потоков.

Классы, управляющие потоком, получают в своих конструкторах уже имеющийся поток и создают новый, преобразованный поток. Можно представлять их себе как "переходное кольцо", после которого идет труба другого диаметра.

Четыре класса созданы специально для преобразования потоков:

FilterReader FilterlnputStream

FilterWriter FilterOutputStream

Сами по себе эти классы бесполезны — они выполняют тождественное преобразование. Их следует расширять, переопределяя методы ввода/вывода. Но для байтовых фильтров есть полезные расширения, которым соответствуют некоторые символьные классы. Перечислим их.

Четыре класса выполняют буферизованный ввод/вывод:

BufferedReader BufferedlnputStream

BufferedWriter BufferedOutputStream

Два класса преобразуют поток байтов, образующих восемь простых типов Java, в эти самые типы:

DatalnputStream DataOutputStream

Два класса содержат методы, позволяющие вернуть несколько символов или байтов во входной поток:

PushbackReader PushbacklnputStream

Два класса связаны с выводом на строчные устройства — экран дисплея, принтер:

PrintWriter PrintStream

Два класса связывают байтовый и символьный потоки:

- inputstreamReader — преобразует входной байтовый поток в символьный поток;
- Outputstreamwriter — преобразует выходной символьный поток в байтовый поток.

Класс streamTokenizer позволяет разобрать входной символьный поток на отдельные элементы (tokens) подобно тому, как класс stringTokenizer, рассмотренный нами в *главе 5,* разбирал строку.

Из управляющих классов выделяется класс sequenceinputstream, сливающий несколько потоков, заданных в конструкторе, в один поток, и класс

LineNumberReader, "умеющий" читать выходной символьный поток построчно. Строки в потоке разделяются символами '\n' и/или '\r'.

Этот обзор классов ввода/вывода немного проясняет положение, но не объясняет, как их использовать. Перейдем к рассмотрению реальных ситуаций.

Консольный ввод/вывод

Для вывода на консоль мы всегда использовали метод printino класса Pnntstream, никогда не определяя экземпляры этого класса. Мы просто использовали статическое поле out класса system, которое является объектом класса PrintStream. Исполняющая система Java связывает это поле с консолью.

Кстати говоря, если вам надоело писать system.out.printino, то вы можете определить новую ссылку на system, out, например:

```
PrintStream pr - System.out;
```

и писать просто pr. printin ().

Консоль является байтовым устройством, и символы Unicode перед выводом на консоль должны быть преобразованы в байты. Для символов Latin 1 с кодами '\u0000' — '\u00FF' при этом просто откидывается нулевой старший байт и выводятся байты '0x00' —'0xFF'. Для кодов кириллицы, которые лежат в диапазоне '\u0400 [1] —'\u04FF [1] кодировки Unicode, и других национальных алфавитов производится преобразование по кодовой таблице, соответствующей установленной на компьютере локали.

Трудности с отображением кириллицы возникают, если вывод на консоль производится в кодировке, отличной от локали. Именно так происходит в русифицированных версиях MS Windows NT/2000. Обычно в них устанавливается локаль с кодовой страницей CP1251, а вывод на консоль происходит в кодировке CP866.

В этом случае надо заменить Printstream, который не может работать с сим- , вольным потоком, на Printwriter и "вставить переходное кольцо" между потоком символов Unicode и потоком байтов system, out, выводимых на консоль, в виде объекта класса OutputstreamWriter. В конструкторе этого объекта следует указать нужную кодировку, в данном случае, CP866. Все это можно сделать одним оператором:

```
PrintWriter pw = new PrintWriter(

new OutputstreamWriter(System.out, "Cp866"), true);
```

Класс Printstream буферизует выходной поток. Второй аргумент true его конструктора вызывает принудительный сброс содержимого буфера в выходной поток после каждого выполнения метода

printin(). Но после print() буфер не сбрасывается! Для сброса буфера после каждого print() надо писать flush(), как это сделано в листинге .2.

Замечание

Методы класса PrintWriter по умолчанию не очищают буфер, а метод print () не очищает его в любом случае. Для очистки буфера используйте метод flush().

После этого можно выводить любой текст методами класса PrintWriter, которые просто дублируют методы класса Printstream, и писать, например,

```
pw.println("Это русский текст");
```

как показано в листинге.1 и на рис..3.

Следует заметить, что конструктор класса PrintWriter, в котором задан байтовый поток, всегда неявно создает объект класса OutputstreamWriter с локальной кодировкой для преобразования байтового потока в символьный поток.

Ввод с консоли производится методами read о класса inputstream с помощью статического поля in класса system. С консоли идет поток байтов, полученных из scan-кодов клавиатуры. Эти байты должны быть преобразованы в символы Unicode такими же кодовыми таблицами, как и при выводе на консоль. Преобразование идет по той же схеме — для правильного ввода кириллицы удобнее всего определить экземпляр класса BufferedReader, используя в качестве "переходного кольца" объект класса inputstreamReader:

```
BufferedReader br = new BufferedReader(

new InputstreamReader(System.an, "Cp866"));
```

Класс BufferedReader переопределяет три метода read о своего суперкласса Reader. Кроме того, он содержит метод readLine ().

Метод readLine о возвращает строку типа string, содержащую символы входного потока, начиная с текущего, и заканчивая символом '\n' и/или '\r'. Эти символы-разделители не входят в возвращаемую строку. Если во входном потоке нет символов, то возвращается null.

В листинге.1 приведена программа, иллюстрирующая перечисленные методы консольного ввода/вывода. На рис. 3 показан вывод этой программы.

Листинг.1. Консольный ввод/вывод

```
import j ava.io.*;

class PrWr{

public static void main(String[] args){

try{

BufferedReader br =

new BufferedReader(new InputstreamReader(System.in, "Cp866"));

PrintWriter pw = new PrintWriter(

new OutputstreamWriter(System.out, "Cp866"), true);
```

```
String s = "Это строка с русским текстом";

System.out.println("System.out puts: " + s);

pw.println("PrintWriter puts: " + s) ;

int c = 0;

pw.println("Посимвольный ввод:");

while((c = br.read()) != -1)

pw.println((char)c);

pw.println("Построчный ввод:");

do{

s = br.readLine();

pw.println(s);

}while(!s.equals("q"));

}catch(Exception e){

System.out.println(e);

}

}

}
```

Поясним рис..3. Первая строка выводится потоком system.out. Как видите, кириллица выводится неправильно. Следующая строка предварительно преобразована в поток байтов, записанных в кодировке CP866.

Затем, после текста "Посимвольный ввод:" с консоли вводятся символы "Россия" и нажимается клавиша <Enter>. Каждый вводимый символ отображается на экране — операционная система работает в режиме так называемого "эха". Фактический ввод с консоли начинается только после нажатия клавиши <Enter>, потому что клавиатурный ввод буферизуется операционной системой. Символы сразу после ввода отображаются по одному на строке. Обратите внимание на две пустые строки после буквы я. Это выведены символы '\n' и '\r', которые попали во входной поток при нажатии клавиши <Enter>. У них нет никакого графического начертания (glyph).

Потом нажата комбинация клавиш <Ctrl>+<Z>. Она отображается на консоль как "^Z" и означает окончание клавиатурного ввода, завершая цикл ввода символов. Коды этих клавиш уже не попадают во входной поток.

Далее, после текста "Построчный ввод:" с клавиатуры набирается строка "Это строка" и, вслед за нажатием клавиши <Enter>, заносится в строку s. Затем строка s выводится обратно на консоль.

Для окончания работы набираем q и нажимаем клавишу <Enter>.

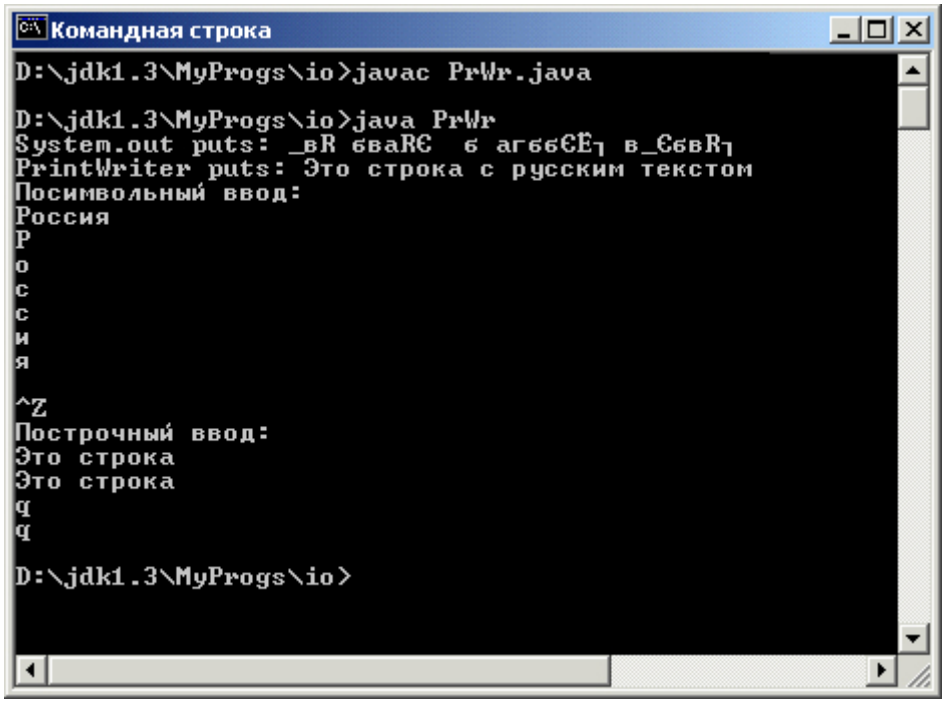

Рис. 3. Консольный ввод/вывод

Файловый ввод/вывод

Поскольку файлы в большинстве современных операционных систем понимаются как последовательность байтов, для файлового ввода/вывода создаются байтовые потоки с помощью классов Fiieinputstream и FiieOutputstream. Это особенно удобно для бинарных файлов, хранящих байт-коды, архивы, изображения, звук.

Но очень много файлов содержат тексты, составленные из символов. Несмотря на то, что символы могут храниться в кодировке Unicode, эти тексты чаще всего записаны в байтовых кодировках. Поэтому и для текстовых файлов можно использовать байтовые потоки. В таком случае со стороны программы придется организовать преобразование байтов в символы и обратно.

Чтобы облегчить это преобразование, в пакет java.io введены классы FineReader и FileWriter. Они организуют преобразование потока: со стороны программы потоки символьные, со стороны файла — байтовые. Это происходит потому, что данные классы расширяют классы InputStreamReader и OutputstreamWriter, соответственно, значит, содержат "переходное кольцо" внутри себя.

Несмотря на различие потоков, использование классов файлового ввода/вывода очень похоже.

В конструкторах всех четырех файловых потоков задается имя файла в виде строки типа string или ссылка на объект класса File. Конструкторы не только создают объект, но и отыскивают файл и открывают его. Например:

```
Fileinputstream fis = new FileInputStreamC'PrWr.Java");

FileReader fr = new FileReader("D:\\jdkI.3\\src\\PrWr.Java");
```

При неудаче выбрасывается исключение класса FileNotFoundException, но конструктор класса FileWriter выбрасывает более общее исключение IOException.

После открытия выходного потока типа FileWriter или FileQutputStEeam содержимое файла, если он был не пуст, стирается. Для того чтобы можно было делать запись в конец файла, и в том

и в другом классе предусмотрен конструктор с двумя аргументами. Если второй аргумент равен true, то происходит дозапись в конец файла, если false, то файл заполняется новой информацией. Например:

```
FileWriter fw = new FileWriter("ch18.txt", true);

FileOutputstream fos = new FileOutputstream("D:\\samples\\newfile.txt");
```

Внимание

Содержимое файла, открытого на запись конструктором с одним аргументом, стирается.

Сразу после выполнения конструктора можно читать файл:

```
fis.read(); fr.read();
```

или записывать в него:

```
fos.write((char)c); fw.write((char)c);
```

По окончании работы с файлом поток следует закрыть методом close ().

Преобразование потоков в классах FileReader и FileWriter выполняется по кодовым таблицам установленной на компьютере локали. Для правильного ввода кирилицы надо применять FileReader, а не FileInputStream. Если файл содержит текст в кодировке, отличной от локальной кодировки, то придется вставлять "переходное кольцо" вручную, как это делалось для консоли, например:

```
InputStreamReader isr = new InputStreamReader(fis, "KOI8_R"));
```

Байтовый поток fis определен выше.

Получение свойств файла

В конструкторах классов файлового ввода/вывода, описанных в предыдущем разделе, указывалось имя файла в виде строки. При этом оставалось неизвестным, существует ли файл, разрешен ли к, нему доступ, какова длина файла.

Получить такие сведения можно от предварительно созданного экземпляра класса File, содержащего сведения о файле. В конструкторе этого класса

```
File(String filename)
```

указывается путь к файлу или каталогу, записанный по правилам операционной системы. В UNIX имена каталогов разделяются наклонной чертой /, в MS Windows — обратной наклонной чертой \, в Apple Macintosh — двоеточием :. Этот символ содержится в системном свойстве file.separator (см. рис. 6.2). Путь к файлу предваряется префиксом. В UNIX это наклонная черта, в MS Windows — буква раздела диска, двоеточие и обратная наклонная черта. Если префикса нет, то путь считается относительным и к нему прибавляется путь к текущему каталогу, который хранится в системном свойстве user.dir.

Конструктор не проверяет, существует ли файл с таким именем, поэтому после создания объекта следует это проверить логическим методом exists ().

Класс File содержит около сорока методов, позволяющих узнать различные свойства файла или каталога.

Прежде всего, логическими методами isFileO, isDirectoryO можно выяснить, является ли путь, указанный в конструкторе, путем к файлу или каталогу.

Для каталога можно получить его содержимое — список имен файлов и подкаталогов— методом list o, возвращающим массив строк stringf]. Можно получить такой же список в виде массива объектов класса File[] методом listFilest). Можно выбрать из списка только некоторые файлы, реализовав интерфейс FileNameFiiter и обратившись к методу

list(FileNameFilter filter).

Если каталог с указанным в конструкторе путем не существует, его можно создать логическим методом mkdir(). Этот метод возвращает true, если каталог удалось создать. Логический метод mkdirso создает еще и все несуществующие каталоги, указанные в пути.

Пустой каталог удаляется методом delete ().

Для файла можно получить его длину в байтах методом length (), время последней модификации в секундах с 1 января 1970 г. методом lastModifiedo. Если файл не существует, эти методы возвращают нуль.

Логические методы canRead (), canwrite () показывают права доступа к файлу.

Файл можно переименовать логическим методом renameTo(Fiie newMame) или удалить логическим методом delete o. Эти методы возвращают true, если операция прошла удачно.

Если файл с указанным в конструкторе путем не существует, его можно создать логическим методом createNewFilet), возвращающим true, если файл не существовал, и его удалось создать, и false, если файл уже существовал.

Статическими методами

createTempFile(String prefix, String suffix, File tmpDir)

createTempFile(String prefix, String suffix)

можно создать временный файл с именем prefix и расширением suffix в каталоге tmpDir или каталоге, указанном в системном свойстве java.io.tmpdir (см. рис. 6.2). Имя prefix должно содержать не менее трех символов. Если suffix = null, то файл получит суффикс .tmp.

Перечисленные методы возвращают ссылку типа File на созданный файл. Если обратиться к методу deieteOnExit (), то по завершении работы JVM временный файл будет уничтожен.

Несколько методов getxxxo возвращают имя файла, имя каталога и другие сведения о пути к файлу. Эти методы полезны в тех случаях, когда ссылка на объект класса File возвращается другими методами и нужны сведения о файле. Наконец, метод toURL () возвращает путь к файлу в форме URL.

В листинге.2 показан пример использования класса File, а на рис..4 — начало вывода этой программы.

Листинг .2. Определение свойств файла и каталога

```
import java.io.*;

class FileTest{

public static void main(String[] args) throws IOException{
```

```
PrintWriter pw = new PrintWriter(
new OutputStreamWriter(System.out, "Cp866"), true);
File f = new File("FileTest.Java");
pw.println();
pw.println("Файл \"" + f.getName() + "\" " +
(f.exists()?"":"не ") + "существует");
pw.println("Вы " + (f.canRead()?"":"не ") + "можете читать файл");
pw.println("Вы " + (f.canWrite()?"":"не ") +
"можете записывать в файл");
pw.println("Длина файла " + f.length() + " б");
pw.println() ;
File d = new File(" D:\\jdkl.3\\MyProgs ");
pw.println("Содержимое каталога:");
if (d.exists() && d.isDirectory()) {
String[] s = d.list();
for (int i = 0; i < s.length; i++)
pw.println(s[i]);
}
}
}
```

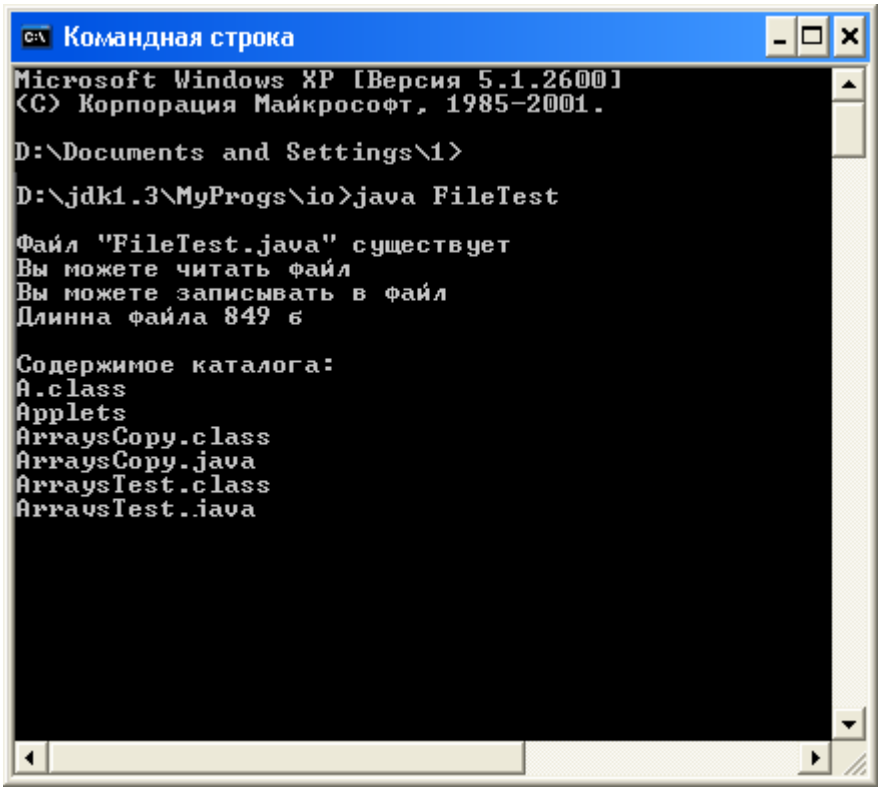

Рис. 4. Свойства файла и начало вывода каталога

Буферизованный ввод/вывод

Операции ввода/вывода по сравнению с операциями в оперативной памяти выполняются очень медленно. Для компенсации в оперативной памяти выделяется некоторая промежуточная область — буфер, в которой постепенно накапливается информация. Когда буфер заполнен, его содержимое быстро переносится процессором, буфер очищается и снова заполняется информацией.

Житейский пример буфера — почтовый ящик, в котором накапливаются письма. Мы бросаем в него письмо и уходим по своим делам, не дожидаясь приезда почтовой машины. Почтовая машина периодически очищает почтовый ящик, перенося сразу большое число писем. Представьте себе город, в котором нет почтовых ящиков, и толпа людей с письмами в руках дожидается приезда почтовой машины.

Классы файлового ввода/вывода не занимаются буферизацией. Для этой цели есть четыре специальных класса BufferedXxx, перечисленных выше. Они присоединяются к потокам ввода/вывода как "переходное кольцо", например:

```
BufferedReader br = new BufferedReader(isr);

BufferedWriter bw = new BufferedWriter(fw);
```

Потоки isr и fw определены выше.

Программа листинга.3 читает текстовый файл, написанный в кодировке CP866, и записывает его содержимое в файл в кодировке KOI8_R. При чтении и записи применяется буферизация. Имя исходного файла задается в командной строке параметром args[0], имя копии — параметром argstl].

Листинг.3. Буферизованный файловый ввод/вывод

```
import java.io.*;

class DOStoUNIX{

public static void main(String[] args) throws IOException{

if (args.length != 2){

System.err.println("Usage: DOStoUNIX Cp866file KOI8_Rfile");

System.exit(0);

}

BufferedReader br = new BufferedReader(

new InputStreamReader(

new FileInputStream(args[0]), "Cp866"));

BufferedWriter bw = new BufferedWriter(

new OutputStreamWriter(

new FileOutputStreamtargs[1]), "KOI8_R"));

int c = 0;

while ((c = br.readO) != -1)

bw.write((char)c);

br.closeO; bw.close();

System.out.println("The job's finished.");

}

}
```

Поток простых типов Java

Класс DataOutputstream позволяет записать данные простых типов Java в выходной поток айтов методами writeBoolean (boolean b), writeBytefint b), writeShort(int h), writeChar(int c), writeInt"(int n), writeLong(long 1), writeFloat(float f), writeDouble(double d).

Кроме того, метод writeBytes(string s) записывает каждый символ строки s в один байт, отбрасывая старший байт кодировки каждого символа Unicode, а метод writecnarststring s) записывает каждый символ строки s в два байта, первый байт — старший байт кодировки Unicode, так же, как это делает метод writeChar ().

Еще один метод writeUTFtstring s) записывает строку s в выходной поток в кодировке UTF-8. Надо пояснить эту кодировку.

Кодировка UTF-8

Запись потока в байтовой кодировке вызывает трудности с использованием национальных символов, запись потока в Unicode увеличивает длину потока в два раза. Кодировка UTF-8 (Universal Transfer Format) является компромиссом. Символ в этой кодировке записывается одним, двумя или тремя байтами.

Символы Unicode из диапазона '\u0000' —'\u007F', в котором лежит английский алфавит, записываются одним байтом, старший байт просто отбрасывается.

Символы Unicode из диапазона '\u0080' —'\u07FF', в котором лежат наиболее распространенные символы национальных алфавитов, записываются двумя байтами следующим образом: символ Unicode с кодировкой 00000xxxxxyyyyyy записывается как 110xxxxx10yyyyyy.

Остальные символы Unicode из диапазона '\u0800' —'\UFFFF' записываются тремя байтами по следующему правилу: символ Unicode с кодировкой xxxxyyyyyyzzzzzz записывается как 1110xxxx10yyyyyy10zzzzzz.

Такой странный способ распределения битов позволяет по первым битам кода узнать, сколько байтов составляет код символа, и правильно отсчитывать символы в потоке.

Так вот, метод writeUTF(string s) сначала записывает в поток в первые два байта потока длину строки s в кодировке UTF-8, а затем символы строки в этой кодировке. Читать эту запись потом следует парным методом readUTF() класса DataInputStream.

Класс DataInputStream преобразует входной поток байтов типа InputStream, составляющих данные простых типов Java, в данные этого типа. Такой поток, как правило, создается методами класса DataOutputstream. Данные из этого потока можно прочитать методами readBoolean(), readByte(), readShort(), readChar(), readInt(), readLong(), readFloat(), readDouble(), возвращающими данные соответствующего типа.

Кроме того, методы readUnsignedByteO H readUnsignedShort () возвращают целое типа int, в котором старшие три или два байта нулевые, а младшие один или два байта заполнены байтами из входного потока.

Метод readUTF(), двойственный методу writeUTF(), возвращает строку типа string, полученную из потока, записанного методом writeUTF ().

Еще один, статический, метод readUTF(Datainput in) делает то же самое со входным потоком in, записанным в кодировке UTF-8. Этот метод можно применять, не создавая объект класса DataInputStream.

Программа в листинге .4 записывает в файл fib.txt числа Фибоначчи, а затем читает этот файл и выводит его содержимое на консоль. Для контроля записываемые в файл числа тоже выводятся на консоль. На рис. .5 показан вывод этой программы.

Листинг.4. Ввод/вывод данных

```
import j ava.io.*;

class DataPrWr{

public static void main(String[] args) throws IOException{

DataOutputstream dos = new DataOutputstream (
```

```
new FileOutputStream("fib.txt"));

int a = 1, b = 1, c = 1;

for (int k = 0; k < 40; k++){

System.out.print(b + " ");

dos.writeInt(b);

a = b; b = c; c = a + b;

}

dos.closet);

System.out.println("\n");

DataInputStream dis = new DataInputStream (

new FileInputStream("fib.txt")) ;

while(true)

try{

a = dis.readInt();

System.out.print(a + " ">;

}catch(IOException e){

dis.close();

System.out.println("End of file");

System.exit (0);

}

}

}
```

Обратите внимание на то, что попытка чтения за концом файла выбрасывает исключение класса IOException, его обработка заключается в закрытии файла и окончании программы.

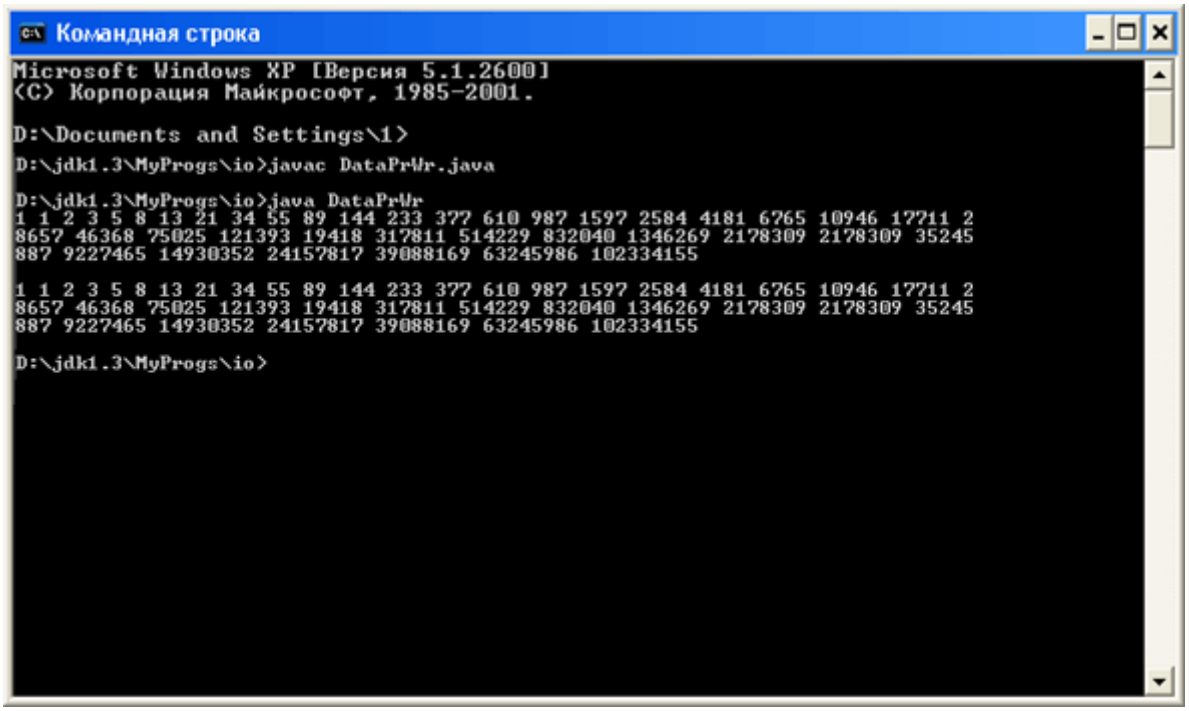

Рис..5. Ввод и вывод данных

Прямой доступ к файлу

Если необходимо интенсивно работать с файлом, записывая в него данные разных типов Java, изменяя их, отыскивая и читая нужную информацию, то лучше всего воспользоваться методами класса RandomAccessFile.

В конструкторах этого класса

RandomAccessFile(File file, String mode)

RandomAccessFile(String fileName, String mode)

вторым аргументом mode задается режим открытия файла. Это может быть строка "r" — открытие файла только для чтения, или "rw" — открытие файла для чтения и записи.

Этот класс собрал все полезные методы работы с файлом. Он содержит все методы классов Datainputstream и DataOutputstream, кроме того, позволяет прочитать сразу целую строку методом readidne () и отыскать нужные данные в файле.

Байты файла нумеруются, начиная с 0, подобно элементам массива. Файл снабжен неявным указателем (file pointer) текущей позиции. Чтение и запись производится, начиная с текущей позиции файла. При открытии файла конструктором указатель стоит на начале файла, в позиции 0. Текущую позицию можно узнать методом getFiiePointer(). Каждое чтение или запись перемещает указатель на длину прочитанного или записанного данного. Всегда можно переместить указатель в новую позицию, роз методом seek (long pos). Метод seek(0) перемещает указатель на начало файла.

В классе нет методов преобразования символов в байты и обратно по кодовым таблицам, поэтому он не приспособлен для работы с кириллицей.

Пример – копирование файлов

```
// Буферизованный файловый ввод/вывод

import java.io.*;

class FtoF{
   public static void main(String[] args) throws IOException{
     if (args.length != 2){
         System.err.println("Usage: FtoF file1 file2");
         System.exit(0);
     }
    int k;
   BufferedReader br = new BufferedReader(
          new InputStreamReader( new FileInputStream(args[0]) )  );

   BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter( new FileOutputStream(args[1]) )  );

   int c = 0;

   while ( (c = br.read()) != -1 )
     {
       bw.write( (char) c );

//       System.out.print( "<"+(char) c  + ">");
//      System.out.print(" = ");
//       System.out.println( (int) c );

     }

   br.close();
   bw.close();

   System.out.println("The job's finished.");

   }

}
```

```
// Буферизованный файловый ввод/вывод с перекодировкой
import java.io.*;
class WINtoKOI8{
   public static void main(String[] args) throws IOException{
     if (args.length != 2){
         System.err.println("Usage: WINtoKIO8 Cp1251-file KOI8_R-file");
         System.exit(0);
     }
    int k;
   BufferedReader br = new BufferedReader(
          new InputStreamReader( new FileInputStream(args[0]), "Cp1251")
);
   BufferedWriter bw = new BufferedWriter(
          new OutputStreamWriter( new FileOutputStream(args[1]), "KOI8_R")
);
   int c = 0;
   while ( (c = br.read()) != -1 )
      bw.write( (char) c );
   br.close();
   bw.close();
   System.out.println("The job's finished.");
   }
}
```