«IO Streams»

Автор: доц. каф. ПО ЭВМ ХНУРЭ Колесников Д.О.

15.1. Виды потоков ввода/вывода

Всего существует 2 вида потоков ввода/вывода:
- байтовые;
- символьные.

Байтовые потоки - последовательность байт (byte).

Символьные - последовательность двухбайтовых символов Unicode (char).

Все потоки ядра Java (стандартного API) являются потомками 4-х суперклассов, которые являются абстрактными и напрямую наследуются от класса Object.

Суперкласс иерархии	*Потоки*
java.io.InputStream	входные байтовые потоки
java.io.OutputStream	выходные байтовые потоки
java.io.Reader	входные символьные потоки
java.io.Writer	выходные символьные потоки

Замечание. В состав API входит класс java.io.RandomAccessFile, который не принадлежит приведенным выше иерархиям, наследуется непосредственно от класса Object и предназначен для работы с файлами, поддерживая произвольный доступ к их содержимому.

Замечание. Большинство потоков ввода/вывода содержаться в пакете java.io, потоки для работы с архивами содержаться в пакете java.util.

15.2. Парные потоки

Предназначение каждого класса-потока заключается в том, чтобы передать или принять последовательность символов или байт.

API Java содержит более 60 потоков, каждый из которых содержит свой собственный набор методов для управлением процессом приема/передачи информации.

Для некоторых потоков существуют парные им в том смысле, что парный поток содержит зеркальное отображение функциональности исходного потока относительно направления передачи информации.

Парные классы в иерархиях *байтовых* потоков	
*Input*Stream	*Output*Stream
ByteArray*Input*Stream	ByteArray*Output*Stream
File*Input*Stream	File*Output*Stream
StringBuffer*Input*Stream	-
Object*Input*Stream	Object*Output*Stream
Filter*Input*Stream	Filter*Output*Stream
Buffered*Input*Stream	Buffered*Output*Stream
-	PrintStream
Zip*Input*Stream	Zip*Output*Stream
Pushback*Input*Stream	-
Data*Input*Stream	Data*Output*Stream

Парные классы в иерархиях *символьных* потоков	
Reader	*Writer*
Buffered*Reader*	Buffered*Writer*
-	Print*Writer*
String*Reader*	String*Writer*
Filter*Reader*	Filter*Writer*
Pushback*Reader*	-
InputStream*Reader*	OutputStream*Writer*
File*Reader*	File*Writer*

15.3. Поле out класса System

Статическое поле *out* класса System имеет тип java.io.PrintStream, который представляет собой надстройку над *байтовым* выходным потоком OutputStream и по умолчанию связан с консольным выводом (дисплеем).

Это, так называемый, поток стандартного вывода.

Программно он может быть надстроен для того, чтобы осуществлять перекодировку символов выводимых данных.

15.4. Классы надстройки

Классы

FilterInputStream, FilterOutputStream;

FilterReader, FilterWriter

являются, соответственно, классами надстройками над классами

InputStream, OutputStream;

Reader и Writer

Суперклассы надстроек являются абстрактными классами.

API Java содержит набор неабстрактных классов-надстроек, которые являются потомками базовых надстроек.

Основное предназначение надстроек - наделение существующего потока новыми свойствами.

Комбинируя исходный поток и классы надстройки, можно создать новый поток с заданным набором свойств.

Если нужно наделить существующий поток некоторым свойством, достаточно надстроить его соответствующим классом надстройкой и работать с объектом последнего.

15.5. Класс DataInputStream

Класс DataInputStream наследует класс надстройку FilterInputStream и позволяет читать данные из входного байтового потока в формате примитивных типов данных: double, boolean и т.д.

Парный класс DataOutputStream наследует класс FilterOutputStream и позволяет записывать значения примитивных типов в выходной байтовый поток, который затем можно будет прочесть используя класс DataInputStream.

Замечание. Экземпляры классов DataInputStream и DataOutputStream надстраивают, соответственно, входной и выходной потоки, которые передаются им как параметры конструкторов при их создании.

15.6. Класс BufferedOutputStream

Класс BufferedOutputStream наследует класс надстройку FilterOutputStream.

Объект этого класса надстраивает выходной байтовый поток и поддерживает буфер определенного размера.

Выходной поток и размер буфера передаются объекту BufferedOutputStream при его создании с помощью конструктора в качестве параметров (размер буфера по умолчанию как правило достаточен для решения большинства возникающих задач).

Парный класс BufferedInputStream наследует надстройку FilterInputStream и надстраивает входной поток, добавляя возможность использовать буфер.

15.7. Класс ByteArrayInputStream

Класс ByteArrayInputStream наследуется напрямую от класса InputStream, при этом байты считываются в массив байт, который передается конструктору объекта класса ByteArrayInputStream при его создании.

Парный класс ByteArrayOutputStream наследуется напрямую от класса OutputStream, при этом байты записываются в массив байт, который передается конструктору объекта класса ByteArrayOutputStream при его создании.

15.8. Класс FileOutputStream

Класс FileOutputStream наследуется напрямую от класса OutputStream и предназначен для записи байт в файл, имя файла передается конструктору при создании объекта.

Парный класс FileInputStream наследуется напрямую от класса InputStream и предназначен для чтения байт из файла, имя файла передается конструктору при создании объекта.

15.9. Класс PushbackInputStream

Класс PushbackInputStream надстраивает входной байтовый поток и позволяет кроме чтения осуществлять запись прочтенных байт обратно во входной поток.

Замечание. Класс PushbackInputStream не имеет парный класс.

Замечание. Существует аналогичный класс для входных символьных потоков.

15.10. Класс RandomAccessFile

Для создания объектов класса RandomAccessFile требуется передать конструктору класса имя файла, с которым предполагается работать, а также обязательно режим доступа к файлу:

'r' (только чтение);
'rw' (чтение и запись).

Вместо имени файла можно передать соответствующий объект класса File.

Замечание. Отсутствует режим доступа "только запись".

15.11. Класс OutputStreamWriter

Класс OutputStreamWriter наследуется от класса Writer, и преобразует выходной символьный поток в выходной байтовый поток. Класс имеет несколько конструкторов, каждый из которых принимает в качестве одного из своих параметров выходной символьный поток.

OutputStreamWriter(OutputStream out)
OutputStreamWriter(OutputStream out, String charsetName)

Второй параметр указывает на кодировку, при этом каждому символу ставится в соответствие совокупность байт, которая является числовым кодом символа в этой кодировке.

Замечание. Если при создании объекта класса OutputStreamWriter используется конструктор без указания кодировки, то конвертирование осуществляется с использованием кодировки по умолчанию.

15.12. Кодировка по умолчанию

При запуске программы кодировку по умолчанию устанавливает JVM в зависимости от операционной системы в которой выполняется программа и ее настроек.

ОС Windows использует в качестве кодировки по умолчанию Windows-1251 (Cp1251), для вывода в консоль используется DOS-кодировка Cp866 (Win OS русской локализации).

15.13. Указание кодировки при компиляции

Для правильного отображения строковых литералов, записанных в программе, следует обеспечить правильное конвертирование этих символов в Unicode при компиляции с помощью javac, указав это при помощи ключа -encoding.

Например, если код программы записан в DOS кодировке Cp866, то компилировать необходимо так:

```
javac –encoding Cp866 NameOfJavaFile
```

15.14. Перекодировка вывода

Все строковые литералы в байт коде классов содержаться в формате Unicode.

При выводе таких строк на экран, в файл и т.д. осуществляется их перекодировка с использованием кодировки по умолчанию.

Например, в ОС Windows кодировкой по умолчанию является Cp1251, поэтому произойдет конвертирование Unicode->Cp1251.

Если вывод осуществляется в консольное окно (с помощью метода System.out.println), то такие строки в общем случае будут отображены неправильно, т.к. Windows для отображения символов в консольном окне использует кодировку Cp866.

Чтобы избежать этого, необходимо явно указать в какой кодировке должны выводится символы.

Достигается это с помощью надстройки стандартного потока вывода.

```
PrintWriter out = new PrintWriter(new
    OutputStreamWriter(System.out, "Cp866"), true);
out.println(s); // вывод на экран строки s в кодировке Cp866
```

Замечание. Второй параметр конструктора PrintWriter указывает на то, что каждый вызов метода println будет принудительно сбрасывать буфер, т.е., после каждого вызова println будет происходить вывод на экран строкового значения параметра этого метода. В противном случае вывод на экран произойдет только тогда, когда буфер принудительно будет сброшен с помощью вызова метода *flush*.

Замечание. Аналогично можно надстроить по сути любой поток, таким образом достигается возможность осуществлять перекодирование символов между любыми двумя допустимыми кодировками.

15.15. Кодировка по умолчанию

Строка с именем кодировки по умолчанию содержится в системном свойстве file.encoding. Свойство можно программно изменить, однако не для всех JDK это приведет к действительной смене кодировки по умолчанию.

```
String encoding, s = "абвгд";
encoding = System.getProperty("file.encoding");
// encoding = Windows-1251
System.setProperty("file.encoding", "Cp866");
encoding = System.getProperty("file.encoding");
// encoding = Cp866
// вывод в кодировке Windows-1251 (jdk1.4/5.0):
System.out.println(s);
```

15.16. Класс InputStreamReader

Класс InputStreamReader наследуется от класса Reader, и преобразует входной байтовый поток в символьный используя заданную кодировку.

InputStreamReader (InputStream in)
InputStreamReader (InputStream in, String charsetName)

15.17. Буферизация

Для ускорения файловых операций чтения/записи следует использовать буферизированные классы: BufferedInputStream и BufferedReader.

```
BufferedReader in1 = new BufferedReader(new
    InputStreamReader(new FileInputStream("file.txt")));

BufferedReader in2 = new BufferedReader(new
    FileReader("file.txt"));

BufferedInputStream in3 = new BufferedInputStream(new
    FileInputStream("file.txt"));
```

15.18. Поле in класса System

Статическое поле in класса System имеет тип InputStream и связано по умолчанию с консольным вводом (клавиатурой). Как правило, приходится надстраивать этот входной поток.

```
BufferedReader in = new BufferedReader(
    new InputStreamReader(System.in));

String s = null;
while (!(s=in.readLine()).equals(""))
    System.out.println(s);
```

15.19. Момент создания файла

Физическое создание файла при помощи класса FileOutputStream происходит до того, как закончится выполняться конструктор, при помощи которого создавался соответствующий объект.

```
File f = new File("file");

// будет создан файл "file"
FileOutputStream out = new FileOutputStream(f);
```

15.20. Класс File

Класс File предназначен для работы с элементами файловой системы (ЭФС).

Представление пути к ЭФС зависит от операционной системы.

Объект класса File представляет собой *абстрактное* представление пути к ЭФС (файлу или каталогу), при этом сам ЭФС может отсутствовать.

При создании объекта класса File всегда задают *абстрактный* путь к ЭФС.

```
public File(String pathname)
```

Путь к ЭФС может быть задан путем сложения двух путей:

```
public File(String parent, String child)
public File(File parent, String child)
```

Абстрактный путь состоит из необязательного системно-зависимого префикса и последовательности имен.
В ОС Windows префиксом является имя устройства (hdd, CD-drive и т.п.), на котором расположена файловая система, \ и \\.

Последовательность имен состоит из совокупности имен каталогов, причем последним именем последовательности может быть имя файла.

Имена разделяются *разделителем*, принятым в ОС в которой выполняется JVM. Сам разделитель хранится в системном свойстве *file.separator*, а также в статическом поле *separator* класса File.

```
File f = new File("folder"+File.separator+"file");
```

Замечание. В ОС Windows разделителем является символ

\ (backslash)

При написании строкового литерала, содержащего путь к ЭФС, следует этот символ удваивать:

```
File f = new File("folder\\file");
```

Также в ОС Windows допускается разделитель

/ (slash).

```
File f = new File("folder/file");
```

15.21. Пустой абстрактный путь

При создании объекта File допускается указывать пустой абстрактный путь.

```
File f = new File("");
```

15.22. Метод getPath

Метод getPath класса File возвращает строковое представление абстрактного пути к ЭФС, который был задан при создании объекта File, при этом используется разделитель файловой системы ОС, в которой выполняется JVM.

Метод *toString* класса File возвращает строку, которую генерирует метод getPath.

```
// если выполняется в Windows
File f2 = new File("folder/file");
// то будет выведено folder\file
System.out.println(f2);
```

15.23. Преобразование абстрактного пути

Абстрактный путь, который задается объекту File при его создании, преобразуется в строку по следующим правилам:

1) разделители за последним именем отбрасываются;

2) несколько подряд идущих разделителей *внутри* пути будут заменены на один;

3) несколько подряд идущих разделителей вначале пути интерпретируются в зависимости от ОС, в Windows они будут заменены на два разделителя.

```
File f = new File("\\/file//");
System.out.print(f); // будет выведено \\file
```

15.24. Метод getAbsolutePath

Метод getAbsolutePath класса File возвращает т.н. *абсолютный* путь к ЭФС вид которого зависит от ОС. В Windows возможно два случая:

1) абстрактный путь не содержит префикса: в этом случае метод вернет строку, состоящую из пути к текущему пользовательскому каталогу, разделителя и абстрактного пути;

2) абстрактный путь содержит префикс, в этом случае метод вернет абстрактный путь.

Замечание. Текущий пользовательский каталог назначается каждой программе при ее выполнении и предназначен для разрешения относительных путей, его имя содержится в системном свойстве *user.dir*

```
String s;
File f = new File("path\\file");

s = f.getAbsolutePath());
// s = System.getProperty("user.dir") + "\\path\\file"

File f2 = new File("Z:\\path/file");
s = f.getAbsolutePath());
// s = "Z:\\path\\file"
```

Замечание. Если абстрактный путь пустой, то будет выведено значение свойства user.dir.

Замечание. Строка, которую возвращает метод getAbsolutePath объекта File, *не зависит* от того, существует или нет ЭФС на который ссылается этот объект.

15.25. Метод listFiles

Метод listFiles класса File возвращает массив объектов File, содержащихся в каталоге, на который указывает объект this.

public File[] listFiles()

Возможны три случая:

1) this ссылается на непустой каталог: возвращаемый массив будет включать объекты File, соответствующие элементам файловой системы, которые содержатся в этом каталоге;

2) this ссылается на пустой каталог: возвращаемый массив будет иметь нулевую длину;

3) this ссылается на файл (не каталог), в этом случае метод listFiles вернет значение null.

Замечание. Вызов метода listFiles желательно предварять вызовом метода isDirectory, который возвращает true в том и только в том случае, когда объект File, на котором он вызывается, ссылается на *существующий каталог*.

```
File[] list;
If (file.isDirectory()) list = file.listFiles();
```

15.26. Интерфейс FileFilter

Метод listFiles может принимать в качестве параметра объект класса, который реализует интерфейс FileFilter. Этот интерфейс содержит один метод accept.

```
boolean accept(File pathname)
```

В результирующий список, который возвращает listFiles будут включены те и только те объекты File, для которых метод accept возвращает true.

```
import java.io.*;

public class Test implements FileFilter {
    public boolean accept(File f) {
        return f.getName().length() < 8;
    }

    public static void main(String[] argv) {
        File path = new File("");
        File[] list = path.listFiles(this);
    }
}

// list содержит только те ЭФС из каталога user.dir
// имена которых имеют длину чьем имени менее 8
символов
```

15.27. Метод getParent

Метод getParent класса File возвращает часть абстрактного пути ЭФС.

Структура абстрактного пути	*Значение возвращаемое getParent*
только одно имя	null
только одно имя, предваренное разделителем	зависит от OS для Windows – разделитель
только одно имя, предваренное двумя разделителями	зависит от OS для Windows – два разделителя
несколько имен	абстрактный путь без последнего имени и разделителя, который его предваряет

15.28. Метод getCanonicalPath

Метод getCanonicalPath класса File возвращает т.н. канонический путь к ЭФС.

Данный метод пытается определить реальный путь к ЭФС в файловой системе ОС, на которой выполняется JVM.

Процесс определения зависит от ОС.

Для Windows он заключается в следующем:

1) с помощью метода getAbsolutePath определяется абсолютный путь

2) соответствующим образом обрабатываются имена «.» (текущий каталог) и «..» (родительский каталог)

3) имя устройства (если оно есть в пути) приводится к верхнему регистру.

```
File f = new File("c:/java/test");
File f2 = new File(f, "..");
String s = f2.getCanonicalPath()); // s = c:\java
```

Практическое задание

1. Прочесть 9-ю главу из [1]
2. По своему номеру в списке сделать задания варианта С за 9-й главой.

[1]: *Java. Промышленное программирование*
И.Н. Блинов, В.С. Романчик
Минск : УниверсалПресс, 2007. — 704 с.