

Программирование на языке Java.

Ввод/Вывод

Картузов А.В.

Обобщенно понятие источника ввода относится к различным способам получения информации: к чтению дискового файла, символов с клавиатуры, либо получению данных из сети. Аналогично, под обобщенным понятием вывода также могут пониматься дисковые файлы, сетевые соединения и т.п. Эти абстракции дают удобную возможность для работы с вводом-выводом (I/O), не требуя при этом, чтобы каждая часть вашего кода понимала разницу между, скажем, клавиатурой и сетью. В Java эта абстракция называется потоком (stream) и реализована в нескольких классах пакета java.io. Ввод инкапсулирован в класс `InputStream`, вывод — в `OutputStream`. В Java есть несколько специализаций этих абстрактных классов, учитывающих их различия при работе с дисковыми файлами, сетевыми соединениями и даже с буферами в памяти.

1. File

`File` — единственный объект в `java.io`, который работает непосредственно с дисковыми файлами. Хотя на использование файлов в приложениях наложены жесткие ограничения, файлы попрежнему остаются основными ресурсами для постоянного хранения и совместного использования информации. Каталог в Java трактуется как обычный файл, но с дополнительным свойством — списком имен файлов, который можно просмотреть с помощью метода `list`.

Замечание:

Java правильно обрабатывает разделители имен каталогов в пути, используемые в UNIX и DOS. Если вы используете стиль UNIX — символы '/', то при работе в Windows Java автоматически преобразует их в '\'. Не забудьте, если вы привыкли к разделителям, принятым в DOS, то есть, к '\', то для того, чтобы включить их в строку пути, необходимо их удвоить, аналогично тому, как это сделано в строке "`\\java\\COPYRIGHT`".

Для определения стандартных свойств объекта в классе File есть много различных методов. Однако, класс File не симметричен. Есть много методов, позволяющих узнать свойства объекта, но соответствующие и функции для изменения этих свойств отсутствуют. В следующем примере используются различные методы, позволяющие и получить характеристики файла:

```
import java.io.File;
class FileTest {
static void p(String s) {
    System.out.println(s);
}
public static void main(String args[]) {
    File f1 = new File("/java/COPYRIGHT");
    p("File Name:" + f1.getName());
    p("Path:" + f1.getPath());
    p("Abs Path:" + f1.getAbsolutePath());
    p("Parent:" + f1.getParent());
    p(f1.exists() ? "exists" : "does not exist");
    p(f1.canWrite() ? "is writeable" : "is not writeable");
    p(f1.canRead() ? "is readable" : "is not readable");
    p("is " + (f1.isDirectory() ? " " : "not") + " a directory");
    p(f1.isFile() ? "is normal file" : "might be a named pipe");
    p(f1.isAbsolute() ? "is absolute" : "is not absolute");
    p("File last modified:" + f1.lastModified());
    p("File size:" + f1.length() + " Bytes");
}
}
```

При запуске этой программы вы получите что-то наподобие вывода:

```
File Name:COPYRIGHT      (имя файла)
Path:/java/COPYRIGHT    (путь)
Abs Path:/Java/COPYRIGHT (путь от корневого каталога)
Parent:/java            (родительский каталог)
exists                  (файл существует)
is writeable            (разрешена запись)
is readable             (разрешено чтение)
is not a directory      (не каталог)
is normal file          (обычный файл)
```

```
is absolute
File last modified:812465204000 (последняя модификация файла)
File size:695 Bytes (размер файла)
```

Существует также несколько классов, использование которых ограничено обычными файлами (их нельзя применять к каталогам). Метод `renameTo(File dest)` применяет файл (нельзя применять файл в другой каталог). Метод `delete` уничтожает дисковый файл. Этот метод может удалять только обычные файлы, каталог, даже пустой, с его помощью удалить не удастся.

1.1. Каталоги

Каталоги — это объекты класса `File`, в которых содержится список других файлов и каталогов. Если `File` ссылается на каталог, то метод `isDirectory` возвращает значение `true`. В этом случае вы можете вызвать метод `list` и извлечь содержимое объекта на файлы и каталоги. В следующем примере показано, как с помощью метода `list` можно просмотреть содержимое каталога.

```
import java.io.File;
class DirList {
public static void main(String args[]) {
    String dirname = "/java"; // имя каталога
    File f1 = new File(dirname);
    if (f1.isDirectory()) { // является ли f1 каталогом
        System.out.println("Directory of " + dirname);
        String s[] = f1.list();
        for (int i=0; i < s.length; i++) {
            File f = new File(dirname + "/" + s[i]);
            if (f.isDirectory()) {
            } else {
                System.out.println(s[i] + " is a file");
            } } } else {
        System.out.println(dirname + " is not a directory");
    } }
}
```

В процессе работы эта программа вывела содержимое каталога `/java` моего персонального компьютера в следующем виде:

```
C:\> java DirList
Directory of /java
bin is a directory
COPYRIGHT is a file
README is a file
```

1.2. Filename Filter

Зачастую у вас будет возникать потребность ограничить количество имен файлов, возвращаемых методом `list`, чтобы получить от него только имена, соответствующие определенному шаблону. Для этого в пакете `java.io` включены интерфейс `FilenameFilter`. Объявите, чтобы реализовать этот интерфейс, три метода: `accept()`, который будет вызываться один раз с каждым новым именем файла. Метод `accept` должен возвращать `true` для тех имен, которые надо включать в список, и `false` для имен, которые следует исключить.

У класса `File` есть два связанных метода, ориентированных на работу с каталогами. Метод `mkdir` создает подкаталог. Для создания каталога, путь к которому не создан, надо использовать метод `mkdirs` — он создаст не только указанный каталог, но и все отсутствующие родительские каталоги.

2. InputStream

`InputStream` — абстрактный класс, задающий используемую в Java модель входных потоков. Все методы этого класса при возникновении ошибки возбуждают исключение `IOException`. Ниже приведен краткий обзор методов класса `InputStream`.

- `read()` возвращает предоставленный очередного доступного символа во входном потоке в виде целого.
- `read(byte b[])` пытается прочесть максимум `b.length` байтов из входного потока в массив `b`. Возвращает количество байтов, в действительности прочитанных из потока.
- `read(byte b[], int off, int len)` пытается прочесть максимум `len` байтов, расположив их в массиве `b`, начиная с элемента `off`. Возвращает количество реально прочитанных байтов.
- `skip(long n)` пытается пропустить во входном потоке `n` байтов. Возвращает количество пропущенных байтов.

- `available()` возвращает количество байтов, доступных для чтения в настоящий момент.
- `close()` закрывает источник ввода. Последующие попытки чтения из этого потока приводят к возбуждению `IOException`.
- `mark(int readlimit)` ставит метку в текущей позиции входного потока, которую можно будет использовать до тех пор, пока из потока не будет прочитано `readlimit` байтов.
- `reset()` возвращает указатель потока на установленную ранее метку.
- `markSupported()` возвращает `true`, если данный поток поддерживает операции `mark/reset`.

3. OutputStream

Как и `InputStream`, `OutputStream` — абстрактный класс. Он задает модель выходных потоков Java. Все методы этого класса имеют тип `void` и возбуждают исключения `IOException` в случае ошибки. Ниже приведен список методов этого класса:

- `write(int b)` записывает один байт в выходной поток. Обратите внимание — аргумент этого метода имеет тип `int`, что позволяет вызывать `write`, предоставляя выражение, при этом не нужно выполнять приведение типа к `byte`.
- `write(byte b[])` записывает в выходной поток весь указанный массив байтов.
- `write(byte b[], int off, int len)` записывает в поток часть массива — `len` байтов, начиная с элемента `b[off]`.
- `flush()` очищает любые выходные буферы, завершая операцию вывода.
- `close()` закрывает выходной поток. Последующие попытки записи в этот поток будут возбуждать `IOException`.

4. Файловые потоки

4.1. FileInputStream

Класс `FileInputStream` используется для ввода данных из файлов. В приведенном ниже примере созданы два объекта этого класса, использующие один и тот же дисковый файл.

```
InputStream f0 = new FileInputStream("/autoexec.bat");  
File f = new File("/autoexec.bat");  
InputStream f1 = new FileInputStream(f);
```

Когда создается объект класса `FileInputStream`, он одновременно с этим открывается для чтения. `FileInputStream` является частью семейства методов абстрактного класса `InputStream`. Попытки применить к объекту этого класса методы `mark` и `reset` приводят к возбуждению исключения `IOException`. В приведенном ниже примере показано, как можно читать одиночные байты, массив байтов и поддиапазон массива байтов. В этом примере также показано, как методом `available` можно узнать, сколько осталось не прочитанных байтов, и как с помощью метода `skip` можно пропустить те байты, которые вы не хотите читать.

```
import java.io.*;  
import java.util.*;  
class FileInputStreamS {  
public static void main(String args[]) throws Exception {  
    int size;  
    InputStream f1 = new FileInputStream("/wwwroot/default.htm");  
    size = f1.available();  
    System.out.println("Total Available Bytes: " + size);  
    System.out.println("First 1/4 of the file: read()");  
    for (int i=0; i < size/4; i++) {  
        System.out.print((char) f1.read());  
    }  
    System.out.println("Total Still Available: " + f1.available());  
    System.out.println("Reading the next 1/8: read(b[])");  
    byte b[] = new byte[size/8];  
    if (f1.read(b) != b.length) {  
        System.err.println("Something bad happened");  
    }  
    String tmpstr = new String(b, 0, 0, b.length);  
    System.out.println(tmpstr);  
    System.out.println("Still Available: " + f1.available());  
    System.out.println("Skipping another 1/4: skip()");  
    f1.skip(size/4);  
    System.out.println("Still Available: " + f1.available());  
    System.out.println("Reading 1/16 into the end of array");  
    if (f1.read(b, b.length-size/16, size/16) != size/16) {
```

```
        System.err.println("Something bad happened");
    }
    System.out.println("Still Available: " + f1.available());
    f1.close();
}
}
```

4.2. FileOutputStream

У класса `FileOutputStream` — два таких же конструктора, что и у `FileInputStream`. Однако, создавать объекты этого класса можно независимо от того, существует ли файл или нет. При создании нового объекта класс `FileOutputStream` предпримет попытку, как открыть файл для вывода, сначала создаст его.

В очередном нашем примере символы, введенные с клавиатуры, считываются из потока `System.in` — по одному символу за вызов, до тех пор, пока не заполнится 12-байтовый буфер. После этого создаются три файла. В первый из них, `file1.txt`, записываются символы из буфера, но не все, а черз один — нулевой, второй и так далее. Во второй, `file2.txt`, записывается весь ввод, попавший в буфер. И наконец в третий файл записывается половина буфера, расположенная в середине, а первая и последняя четверти буфера не выводятся.

```
import java.io.*;
class FileOutputStreamS {
public static byte getlnput() [] throws Exception {
    byte buffer[] = new byte[12];
    for (int i=0; i<12; i++) {
        buffer[i] = (byte) System.in.read();
    }
    return buffer;
}
public static void main(String args[]) throws Exception {
    byte buf[] = getlnput();
    OutputStream f0 = new FileOutputStream("file1.txt");
    OutputStream f1 = new FileOutputStream("file2.txt");
    OutputStream f2 = new FileOutputStream("file3.txt");
    for (int i=0; i < 12; i += 2) {
        f0.write(buf[i]);
    }
}
```

```
    }  
    f0.close();  
    f1.write(buf);  
    f1.close();  
    f2.write(buf, 12/4, 12/2);  
    f2.close();  
    }  
}
```

Замечание:

В настоящее время не существует способа открыть `FileOutputStream` для дозаписи в конец файла. Если вы открываете файл с помощью конструктора `FileOutputStream`, прежнее содержимое этого файла теряется. Это—явный недостаток реализации Java.

5. `ByteArrayInputStream`

`ByteArrayInputStream`—это реализация входного потока, в котором в качестве источника используются массив типа `byte`. У этого класса два конструктора, каждый из которых в качестве первого параметра принимает байтовый массив. В приведенном ниже примере создаются два объекта этого типа. Эти объекты инициализируются символами латинского алфавита.

```
String tmp = "abcdefghijklmnopqrstuvwxyz";  
byte b[] = new byte [tmp.length()];  
tmp.getBytes(0, tmp.length(), b, 0);  
ByteArrayInputStream input1 = new ByteArrayInputStream(b);  
ByteArrayInputStream input2 = new ByteArrayInputStream(b, 0, 3);
```

6. `ByteArrayOutputStream`

У класса `ByteArrayOutputStream` — два конструктора. Первая форма конструктора создает буфер размером 32 байта. При использовании второй формы создается буфер с размером, заданным параметром конструктора (в приведенном ниже примере — 1024 байта):

```
OutputStream out0 = new ByteArrayOutputStream();  
OutputStream out1 = new ByteArrayOutputStream(1024);
```


Программирование на языке Java. Ввод/Вывод

В следующем примере объект `ByteArrayOutputStream` заполняется символами, введенными с клавиатуры, после чего с ним выполняются различные манипуляции.

```
import java.io.*;
import java.util.*;
class ByteArrayOutputStreamS {
public static void main(String args[]) throws Exception {
    int i;
    ByteArrayOutputStream f0 = new ByteArrayOutputStream(12);
    System.out.println("Enter 10 characters and a return");
    while (f0.size() != 10) {
        f0.write( System.in.read());
    }
    System.out.println("Buffer as a string");
    System.out.println(f0.toString());
    System.out.println ("Into array");
    byte b[] = f0.toByteArray();
    for (i=0; i < b.length; i++) {
        System.out.print((char) b[i]);
    }
    System.out.println();
    System.out.println("To an OutputStream()");
    OutputStream f2 = new FileOutputStream("test.txt");
    f0.writeTo(f2);
    System.out.println("Doing a reset");
    f0.reset();
    System.out.println("Enter 10 characters and a return");
    while (f0.size() != 10) {
        f0.write (System.in.read());
    }
    System.out.println("Done.");
}
}
```

Заглянув в созданный в этом примере файл `test.txt`, мы увидим там именно то, что ожидали:

```
C:\> type test.txt
```

```
0123456789
```

StringBufferInputStream

StringBufferInputStream идентичен классу ByteArrayInputStream с тем исключением, что внутренним буфером объекта этого класса является экземпляр String, а не байтовый массив. Кроме того, в Java нет соответствующего классу StringBufferOutputStream. У этого класса есть единственный конструктор:

```
StringBufferInputStream( String s)
```

7. Фильтруемые потоки

При работе системы вывода в среде с параллельными процессами при отсутствии синхронизации могут возникать неожиданные результаты. Причиной этого являются попытки различных подпроцессов одновременно обратиться к одному и тому же потоку. Все конструкторы и методы, имеющиеся в этом классе, идентичны тем, которые есть в классах InputStream и OutputStream, единственно отличия классов фильтруемых потоков в том, что их методы синхронизованы.

8. Буферизованные потоки

Буферизованные потоки являются расширенными классами фильтруемых потоков, в них к потокам ввода-вывода присоединяется буфер в памяти. Этот буфер выполняет две основные функции:

- Она дает возможность исполняющей среде Java проделывать за один раз операции ввода-вывода с большим количеством байтов, тем самым повышая производительность среды.
- Поскольку у потока есть буфер, становятся возможными такие операции, как пропуск данных в поток, установка потока и очистка буфера.

8.1. BufferedInputStream

Буферизация ввода-вывода — общепринятый способ оптимизации таких операций. Класс BufferedInputStream в Java дает возможность “окружить” любой объект InputStream буферизованным потоком, и, тем самым, получить выигрыш в производительности. У этого класса два конструктора. Первый из них

```
BufferedInputStream(InputStream in)
```

создаёт буферизованный поток, используя для него буфер длиной 32 байта. Во втором

```
BufferedInputStream(InputStream in, int size)
```

размер буфера для создаваемого потока задаётся вторым параметром конструктора. В обоих случаях оптимальный размер буфера зависит от операционной системы, количества доступной оперативной памяти и конфигурации компьютера.

8.2. BufferedOutputStream

Вывод в объект `BufferedOutputStream` идентичен выводу в любой `OutputStream` с той разницей, что новый подкласс содержит дополнительный метод `flush`, применение для принудительной очистки буфера и физического вывода на внешнее устройство хранящейся в нём информации. Первая форма конструктора этого класса:

```
BufferedOutputStream(OutputStream out)
```

создаёт поток с буфером размером 32 байта. Вторая форма:

```
BufferedOutputStream(OutputStream out, int size)
```

позволяет задавать требуемый размер буфера.

8.3. PushbackInputStream

Одно из не обычных применений буферизации — реализация операции `pushback` (вернуть назад). `Pushback` применяется к `InputStream` для того, чтобы послепрочтенный символ вернуть обратно во входной поток. Однако возможности класса `PushbackInputStream` весьма ограничены—любая попытка вернуть в поток больше одного символа приведёт к немедленному возбуждению исключения `IOException`. У этого класса — единственный конструктор

```
PushbackInputStream(InputStream in)
```

Помимо уже хорошо нам знакомых методов класса `InputStream`, `PushbackInputStream` содержит метод `unread(int ch)`, который возвращает заданный аргументом символ `ch` во

входной поток.

9. SequenceInputStream

Класс `SequenceInputStream` предоставляет новую возможность слияния нескольких входных потоков в один. В конструктор класса `SequenceInputStream` в качестве параметра используются либо два объекта `InputStream`, либо перечисление объектов `InputStream`:

```
SequenceInputStream(Enumeration e) SequenceInputStream(InputStream s0, InputStream s1)
```

В процессе работы класс выполняет поступающие запросы, считывая информацию из первого входного потока до тех пор, пока он не закончится, после чего переходит ко второму и т.д.

10. PrintStream

Класс `PrintStream` предоставляет все инструменты форматирования, которые мы использовали в примерах для вывода через файлы и скрипты пакета `System` с самого начала книги. Вы уже привыкли писать “`System.out.println`”, но сильно задумываясь при этом о тех классах, которые занимаются форматированием выводимой информации. У класса `PrintStream` два конструктора: `PrintStream(OutputStream out)` и `PrintStream(OutputStream out, boolean autoflush)`. Параметр `autoflush` второго из них указывает, должна ли исполняющая среда Java автоматически выполнять операцию очистки буфера над выходным потоком.

В Java-объектах `PrintStream` есть методы `print` и `println`, “умеющие” работать с любыми типами данных, включая `Object`. Если в качестве аргумента этих методов используется не один из примитивных типов, то они вызывают метод `toString` класса `Object`, после чего выводят полученный результат.

Замечание:

В настоящее время в Java отсутствуют средства для форматирования выводимых данных простых типов, например, типов `int` и `float`. В C++ предусмотрены функции для форматирования чисел с плавающей точкой, позволяющие, например, задать вид вывода, при котором в напечатанном числе будет четыре цифры до десятичной точки и три—после.

11. По течению грести легче

Потоки в Java предоставляют программисту ясную абстракцию для выполнения сложных и зачастую громоздких операций ввода-вывода данных. Java-программы, опирающиеся на абстракции высокого уровня—классы `InputStream` и `OutputStream`, будут и в будущем функционировать правильно—даже тогда, когда будут изобретены новые улучшенные реализации классов ввода-вывода. Как вы увидите в следующих главах, такая модель прекрасно работает и при переходе от набора потоков, ориентированных на файловую систему, к работоспособным потокам и сокетами.