Начинаем программировать на языке Java. Часть 3

Дмитрий Рамодин

Вот мы и подошли вплотную к важнейшему этапу — созданию элементов ввода/вывода. Для этого язык Java предлагает целую гамму классов потоков ввода/вывода. Именно их мы и рассмотрим на этом занятии.

Для языков Си++ и Java характерно использование потоков ввода/вывода вместо файлов, как это делалось ранее. Поток ввода/вывода — это некоторый условный канал, по которому отсылаются и получаются данные. При этом совершенно не важно, что стоит за конкретным потоком: файл, блок памяти, экран и т. д. С точки зрения программиста, поток представляет собой ленточный транспортер, на который можно последовательно помещать куски данных, а лента доставит их по назначению. Остальные детали реализации не важны. Такая концепция помогает унифицировать методы работы со всеми устройствами ввода/вывода, сводя все к методам открытия потока, его закрытия, чтения данных из потока и запись данных в него. Конечно же, существуют исключения, но в целом разработчики языка Java старались соблюдать условия унифицированного интерфейса управления потоком. Во всю эту концепцию не вписываются лишь два класса: File и FileDescriptor. Первый осуществляет системные операции, как-то: создание, удаление, переименование файлов, и часто служит промежуточным звеном при использовании потоков. А вот второй, FileDescriptor, — это совсем отдельный случай. Он хранит три дескриптора стандартных файловых потоков (ввода, вывода и сообщений об ошибке) и содержит средства, позволяющие проверить правильность любого дескриптора файла.

1. InputStream и OutputStream

Два класса, InputStream и OutputStream, из упаковки java.io служат предками для

большинства классов потоков ввода/вывода языка Java, поэтому понимание их структуры и возможностей важно для программиста.

Абстрактный класс InputStream предоставляет начальный интерфейс к потоку ввода данных и частично реализует его. С помощью набора методов, реализуемого классом InputStream, можно читать байты или массивы байтов, узнавать количество доступных для чтения данных, отмечать место в потоке, где в настоящий момент происходит чтение, сбрасывать указатель текущей позиции в потоке и пропускать ненужные байты в потоке.

Открыть поток ввода можно, создав объект класса InputStream. Закрыть его можно двумя способами: дождаться, пока сборщик мусора (garbage collector) Java будет искать в памяти компьютера неиспользуемые классы и закроет ваш поток, или же закрыть его методом close(), как обычно и делается.

Для создания потоков ввода применяется другой класс — OutputStream, который, как и InputStream, является абстрактным. Методы, предоставляемые OutputStream, позволяют записывать байты и массивы байтов в поток вывода. Как и InputStream, поток OutputStream открывается, когда вы его создаете, и закрывается либо сборщиком мусора, либо методом close().

На базе двух упомянутых выше классов InputStream и OutputStream наследуются несколько классов с конкретной спецификой применения, как, например, классы FileInputStream и FileOutputStream для записи и чтения данных из файла.

2. FileInputStream и FileOutputStream

Большинству программистов не просто оперировать сравнительно новым понятием "потоки". Однако два класса — FileInputStream и FileOutputStream — обычно понятны всем, поскольку это ни что иное, как потоки ввода из файла и вывода в него. Мы начнем с рассмотрения FileInputStream. Это довольно универсальный класс, открывающий поток ввода по имени файла. Замечательной особенностью этого класса можно считать возможность создания потока ввода данных по объекту класса File и файловому дескриптору FileDescriptor. Вот, оказывается, какое у этих двух классов применение!

Второй класс, FileOutputStream, служит для записи данных в файл и во многом схож с FileInputStream. Объекты класса FileOutputStream также создаются по имени файла или по объектам File или FileDescriptor. Вот так выглядит простейшая программа на языке Java, копирующая содержимое одного файла в другой файл:

```
import java.io.*;
class CopyFile
{
public static void main (String[] args)
 {
   try
   {
    File inFile = new
    File("infile.dat");
    File outFile = new
    File("outfile.dat");
    FileInputStream inStream = new
    FileInputStream(inFile);
    FileOutputStream outStream = new
    FileOutputStream(outFile);

    int c;
    while ((c = inStream.read()) != -1)
    { outStream.write(c); }
       inStream.close();
       outStream.close();
   } catch (FileNotFoundException ex) {}
 }
}
```

В данном случае мы использовали метод создания потоков с промежуточными объектами класса File как пример использования, но ничего не мешает сделать это более простым способом:

```
    FileInputStream inStream =
    new FileInputStream("infile.dat");
    FileOutputStream outStream =
    new FileOutputStream("outfile.dat");
```

Результат будет одним и тем же — будут созданы файловые потоки infile.dat и outfile.dat.

3. PipedInputStream и PipedOutputStream

Интересное применение могут найти специализированные потоковые классы. Так, например, два класса, PipedInputStream и PipedOutputStream, введены в иерархию классов Java для создания каналов (pipes), передачи данных от одной программы к другой или от одного потока выполнения (thread) к другому. Каналы широко используются в операционных системах UNIX.

Каналы удобны как средство переопределения потоков, как это делается операторами ">", ">>" или "<" операционной системы для переназначения ввода и вывода данных для программы. В Java создание такого канала сводится к двум строкам исходного текста:

```
outPipe = new PipedOutputStream ();
inPipe = new PipedInputStream(outPipe);
```

4. SequenceInputStream

Если необходимо объединить в один поток данные из нескольких потоков, на помощь придет класс SequenceInputStream. Он очень прост в использовании: достаточно передать ему список файлов, выполненных в виде класса, унаследованного от интерфейса Enumeration. Задача значительно упрощается, если требуется объединить всего два потока. Создайте объект класса SequenceInputStream, вызвав другой его конструктор, принимающий два аргумента типа InputStream. Для примера создадим класс списка файлов и передадим его объекту класса SequenceInputStream:

```
import ileLjava.util.*;
import java.io.*;
class FileList implements Enumeration
{
  String[] fist;
  int count = 0;
   FileList (String[] listOfFiles)
```

```
   { this.fileList = listOfFiles;}
    public boolean hasMoreElements()
   {
         if (current < fileList.length)
                 return true;
         else return false;
   }
   public Object nextElement()
   {
         InputStream is = null;
         if (!hasMoreElements())
                 throw new
                 NoSuchElementException
                 ("No more files.");
         else
         {
                 String nextElement =
                 fileList[current];
                 current++;
                 is = new
                 FileInputStream
                 (nextElement);
         }
 return is;
   }
}
```

Теперь, когда в нашем распоряжении имеется класс-список, на его основе можно создать единый поток данных из нескольких отдельных потоков:

```
import java.io.*;
class Example
{
   public static void main
   (String[] args)
   {
    ListOfFiles mylist
    = new ListOfFiles(args);
    SequenceInputStreamis
    = new SequenceInputStream(mylist);
```

```
   int c;
// Здесь производятся некоторые
    действия над полученным потоком
     s.close();
   }
}
```

5. DataInputStream и DataOutputStream

DataInputStream и DataOutputStream относятся к так называемым фильтровым классам, то есть классам, задающим фильтры для чтения и записи определенных форматов данных. Фильтровые классы не работают сами по себе, а принимают или отсылают данные простым потокам FileInputStream, FileOutputStream и т. д. Обычное создание потока вывода данных на базе класса DataOutputStream сводится к одной строке:

```
DataOutStream is =
     new DataOutStream
     ( new FileOutputStream ( "data.dat" ));
```

После того как поток создан, в него можно выводить форматированные данные. Для этого в арсенале класса DataOutputStream имеется целый набор методов writeXXX() для записи различных данных, где XXX — название типа данных. Вот так выглядит фрагмент кода для вывода в созданный нами поток data.dat:

```
dos.writeDouble(doubleVar);
dos.writeInt(intVar);
dos.writeChars(StringVar);
dos.close();
```

Мне кажется, комментарии излишни, поскольку имена методов сами говорят о том, какой тип данных они выводят.

Ну а теперь проверим, как записались наши данные в data.dat, и заодно посмотрим, какие методы для чтения имеются в фильтровом потоке ввода данных DataInputStream:

```
DataInputStream dis =
     new DataInputStream (
     new FileInputStream ("data.dat" ));
doubleVar = dis.readDouble();
```

```
intVar = dis.readInt();
StringVar = dis.readLine();
dis.close();
```

Как видно из примера, методы чтения readXXX() класса DataInputStream практически полностью соответствуют методам writeXXX() класса DataOutputStream, за исключением методов writeChars и readLine, имеющим по неясным мне причинам различные названия.

6. RandomAccessFile

Класс произвольного доступа к файлу RandomAccessFile может реализовывать интерфейсы как к DataInput, так и к DataOutput. Это означает, что класс RandomAccessFile может быть использован как для ввода данных из файла, так и для вывода в файл.

Для того чтобы создать объект класса RandomAccessFile, необходимо вызвать его конструктор с двумя параметрами: именем файла для ввода/вывода и режимом доступа к открываемому файлу. Так может выглядеть открытие файла для чтения информации:

```
new RandomAccessFile
    ("some.dat", "r");
```

А в следующем примере файл открывается как для чтения, так и для записи:

```
new RandomAccessFile
     ("some.dat", "rw");
```

После того как файл открыт, вы можете использовать любые методы readXXX() и writeXXX() для ввода и вывода.

Основным же преимуществом класса RandomAccessFile является его способность читать и записывать данные в произвольное место файла. Программисты, работающие на Си и Си++, легко обнаружат, что в основе управления файлом лежит уже знакомый им файловый указатель, отмечающий текущую позицию, где происходит чтение или запись данных. В момент создания объекта класса RandomAccessFile файловый указатель устанавливается в начало файла и имеет значение 0. Вызовы методов readXXX() и writeXXX() обновляют позицию файлового указателя, сдвигая его на

количество прочитанных (записанных) байтов. Для произвольного сдвига файлового указателя на некоторое количество байтов можно применить метод skipBytes(), или же установить файловый указатель в определенное место файла вызовом метода seek(). Для того чтобы узнать текущую позицию, в которой находится файловый указатель, нужно вызвать метод getFilePointer().

Помимо классов потоков ввода/вывода, описанных на этом занятии, существуют еще несколько классов, о которых мы не сказали. Они не так часто употребляются в программах Java, и поэтому вы можете отыскать их и освоить самостоятельно в тот момент, когда они вам понадобятся.

Мир ПК #02/97