Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ
Ведущий
методист колледжа
_____Е.В. Паскал
«___»_____2021

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Учебная дисциплина: «Основы кроссплатформенного программирования»

Лабораторная работа № 8
Инструкционно-технологическая карта

Тема: «Ввод-вывод в Java программах. Файловый ввод-вывод»

Цель: Научиться работать с потоками ввода-вывода, в частности InputStream и OutputStream.

Время выполнения: 4 часа

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретические сведения;
2. Ответить на контрольные вопросы;
3. Откомпилировать примеры программ из раздела «Теоретические сведения»;
4. Выполнить ИДЗ.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ДЛЯ ВЫПОЛНЕНИЯ РАБОТЫ

Практически в каждом языке программирования есть краеугольный камень, вызывающий у разработчиков бурные дискуссии, наиболее сложной, постоянно дорабатываемой системы ввода/вывода. Для реализации ввода/вывода создано множество подходов, однако оптимум, так и не достигнут. Основная трудность в реализации является создание кода, охватывающего все задачи ввода/вывода, реализация всех возможностей. Необходимо реализовать множество вариантов доступа к еще большему количеству видов вывода. Исходный код системы ввода/вывода должен уметь работать с такими видами ввода/вывода, как сетевое соединение, консоль, файлы. К видам потока должно быть реализовано минимум 7-10 способов доступа в случайном порядке, последовательный доступ, бинарный доступ, пословный, построчный, посимвольный, буферный доступ.

Концепция ввода/вывода

Чтобы реализовать такую задачу, разработчики создали множество классов для системы ввода/вывода byte-ориентированных и char-ориентированных данных в формате Unicode. На самом деле, создано такое количество классов и методов вывода, что в них действительно можно запутаться. Задача темы разобраться и понимать, зачем в стандартной библиотеке вода/вывода такое количество классов. Для реализации ввода/вывода в стандартной библиотеке используют концепцию потока. Абстракцию потока представляют в виде источника данных, объекта способного принимать или производить данные порциями. Абстракцию потока создают для сокрытия механизма реализации, того как происходит внутренняя обработка данных, а также, как внутри устройства производится ввод/вывод данных. Все библиотечные классы разделяются на классы вывода и ввода. Среди этих классов есть иерархия, согласно которой все классы, созданные из классов Reader или InputStream наследуют метод read( ), предназначенный для чтения единичного байта или чтения массива байт. Аналогично методу чтения, в каждом классе, наследнике классов Writer или OutputStream, существует метод записи write( ), предназначенный для записи единичного байта или для записи массива байт. Первая особенность, эти методы практически не приспособлены к использованию, они необходимы для создания классов наследников, их полезных интерфейсов. В результате, библиотечные полезные классы помогут создавать пользовательский объект потока и предоставят множество методов, объектов для реализации функциональности. По сути, для реализации потока в Java создаются несколько объектов, создающие необходимый функционал, и этот факт запутывает будущих программистов.

Для удобства понимания, все классы, работающие на вывод, наследуются из класса OutputStream. Под понятием вывод подразумевается вывод, передача информации сетевому соединению, в файл, на флешку, на экран монитора.

Все классы, работающие с вводом, наследуются из класса-прародителя InputStream. Потоком ввода может быть разный источник, всевозможные способы получения информации. Информацию в Java можно считывать из сети, из клавиатуры, из файлов, расположенных на жестком диске, съемных флеш устройств.

Такая реализация позволила создавать работающие с любым устройством ввода вывода программы, код которых практически не изменяется от перемены источника информации. В этом заслуга применяемой абстракции (stream) поток. Код реализации класса содержится в нескольких пакетах стандартной библиотеки java.io, при этом код вывода данных инкапсулирован в класс OutputStream, а ввод инкапсулирован в класс InputStream. Такая реализация с помощью абстракции помогает писать программный код, работающий со всеми устройствами, будь-то буфер памяти, сеть или клавиатура.

Для удобства работы, организации логики, в языке существует несколько специализаций данных абстрактных классов, взаимодействующие с файлами на диске, с памятью (буферы) либо по сети.

Типы Stream

Работа OutputStream/InputStream заключается в представлении пользователю классов, которые осуществляют вывод данных из различных источников, ввод данных в разные источники. Все источники можно разделить на виды:

1. Файл (File) – работа с данными, содержащимися на носителе информации.
2. Массив байт.
3. Объект String.
4. Fifo, как и в физической трубе, элемент, помещенный с одной стороны выводиться с другой (первый зашел, первый вышел).
5. Последовательность потоков, которые можно собрать в единый поток.
6. Всевозможные источники на подобии Internet соединений.

File (Файл)

В стандартной библиотеке ввода/вывода объект File осуществляет обмен данными с дисковыми файлами. Так как Java разрабатывался для сайтов в интернете, применение объектов, работающих с файлами, в апплетах жестко ограничиваются, но файлы, по-прежнему являются основными ресурсами совместного использования данных, информации, для ее постоянного хранения.

В Java особенная система хранения каталогов. Они представляются в виде файла с дополнительным параметром, в виде свойств и списка файлов, их имен, содержащиеся в каталоге. Чтобы получить список файлов, достаточно вызвать метод list. Указывать путь к каталогу достаточно просто, как это делается в ОС (в DOS, Windows '\', Unix - '/'), с единственным отличием, черту '\' в пути необходимо удваивать: \\data\\lib. Класс File не симметричен, так как в нем методов, определяющих, задающих стандартные свойства каждого объекта этого типа значительно меньше, чем методов, позволяющих узнать определенные свойства объектов. В примере приведены всевозможные методы для получения характеристик, свойств файла, при этом, аналогичных функций для изменения соответствующих характеристик нет.

```
import java.io.File;
class EXFile {
static void prn(String arg) { System.out.println(arg);}

public static void main(String arg[]) {
File fileobj = new File("\\Data\\lib");
prn("Name of file:" + fileobj.getName());
prn("Size of file:" + fileobj.length() + " Bytes");
prn("Last modified:" + fileobj. lastModified());
```

```
prn("Parent:" + fileobj.getParent());
prn("Path:" + fileobj.getPath());
prn("Abs Path:" + fileobj.getAbsolutePath());
prn("File " + (fileobj.exists() ? "exists" : " not exist"));
prn(fileobj.canRead() ? "can read" : "can't read");
prn(fileobj.canWrite() ? "can write" : "can't write");
prn("This is " + (fileobj.isDirectory() ? " " : "not") + " directory");
prn("This is "+ (fileobj.isAbsolute() ? "absolute" : " not absolute")  +
   (fileobj.isFile() ? " normal file" : " a named pipe"));

} }
```

Результат исполнения программы:

Name of file: lib (имя файла)
Size of file: 8790 Bytes (размер файла)
Last modified:821456789000 (последняя модификация файла)
Parent:/data (родительский каталог)
Path:/data/lib (путь)
Abs Path:/dat/lib (путь, начиная с корневого каталога)
File exists (файл существует)
Can read(разрешено чтение)
Can write(разрешена запись)
This is not a directory (этот объект не каталог)
This is absolute normal file (обычный файл)

Кроме методов для получения свойств файла, есть сервисные методы, действие которых ограничено влиянием на обычные файлы, при этом не работают с каталогами. Метод для переименования файлов renameTo(File dest) не позволяет переносить файлы в другой каталог. Для уничтожения файла используется метод delete(File arg). Метод не позволяет удалять каталоги, даже пустые, только файлы.

Каталог

Такие объекты класса File отличаются от обычных объектов file тем, что содержат список других каталогов и файлов. Если результат вызова метода isDirectory() возвращает true, то объект File, ссылается на каталог, соответственно можно применить к объекту метод list() и получить его содержимое, список имен каталогов и файлов. Пример использования методов для каталогов.

```
import java.io.File; //считывание классов File  из пакета java.io
class DL {
void public static main(String arg[])
{
String name = "/data"; // имя каталога data
File fileobj = new File(name);
if (fileobj.isDirectory()) { // если fileobj является каталогом, то вывод сообщения
System.out.println("This directory of ' + name);
String s[] = fileobj.list();
for ( int cnt=0; cnt < s.length; cnt++) {
File fileobj2 = new File(name + "/" + s[i]);
```

```
// является ли fileobj2 каталогом, если да то выводится сообщение?
System.out.println(s[i] + (fileobj2.isDirectory()? " is a directory":" is a file"));
            }
//Выводится сообщение, в случае, если указана не директория
   } else System.out.println(name + " it's not a directory");
 }
}
```

Результат работы программы, является отображение содержимого каталога /data:

```
C:\> java DL
Directory of /data
library is a directory
lib is a file
help is a file
```

FilenameFilter

В процессе использования выборок имен файлов и каталогов, содержащихся в папке, при их огромном количестве, может потребоваться ограничение списка имен, который возвращает метод list(). Иными словами нужен фильтр имен в соответствии с шаблоном. Для фильтрации имен в стандартную библиотеку пакета java.io, включен стандартный интерфейс FilenameFilter. Для реализации интерфейса, достаточно объекту определить метод accept(). Метод accept(), вызывается для каждого имени файла, содержащегося в папке. Результат работы метода, логическое «да», в случае, если имя соответствует или «ложь», если имя не нужно включать в итоговый список.

Немного сложно реализована работа с каталогами. В классе File метод mkdirs() применяется для создания каталога и всех родительских каталогов, фактически создает сразу ветку. Второй метод mkdir() применяется в тех случаях, когда необходимо создать подкаталог.

InputStream

Первичный абстрактный класс InputStream скрывает реализацию, а также определяет модель поведения входных потоков в Java, на основании этого класса. Любая ошибка, возникшая в методах этого класса, генерируют исключение IOException, что удобно для перехвата и обработки. Каждый источник данных имеет ассоциированный класс InputStream. Класс InputStream содержит следующие методы:

- Метод read() - возвращает значение целого типа очередного символа, доступного во входном потоке;
- Метод read(byte arr[]) - метод для массового считывания данных, который считывает максимум байтов (не более arr.length) из потока, входящих данных в аргумент массива arr и возвращает фактическое количество байтов, считанных из потока;
- Метод read(byte arr[], int start, int len) считывает из входного потока максимум len байтов, начиная с элемента start в массив arr типа байт (если в

потоке было меньше данных, то вернется меньшее количество байтов, а если больше, то вернется не более len байтов);

- Метод skip(long n) служит для пропуска n байтов из входного потока и результатом вызова метода, будет являться n фактически пропущенных байтов;
- Метод available () возвращает фактическое количество байтов, которые доступны для считывания на момент вызова метода;
- Метод close() предназначен для закрытия источника ввода. После исполнения этого метода, поток закрывается и любые обращения к его данным, попытки чтения данных из потока приводят к генерированию IOException;
- Метод mark(int readlimit) предназначен для задания метки в текущую позицию потока входящих данных. Передаваемый аргумент задает пограничное значение считываемых байт, после превышения, которых не позволительно использовать заданную метку. Иными словами, метка актуальна до тех пор, пока не будет прочитано из потока readlimit байтов;
- Метод reset() предназначен для возврата указателя потока на точку где была установлена метка;
- Метод marksupported() необходим для проверки свойств потока. Если поток поддерживает операцию mark/reset, то метод возвращает логическое true.

OutputStream

Первичный абстрактный класс OutputStream аналогичен классу InputStream, он скрывает реализацию, а также определяет модель поведения объектов входных потоков, созданных на основании этого класса. Внутри класса все методы типа void, и в случае возникновения ошибки, генерируют исключение IOException. Класс OutputStream содержит следующие методы:

- Метод write(int b) предназначен для записи единичного байта в поток выходных данных. Для удобства аргумент метода задан типом int. Такой синтаксис позволяет вызывать метод write с данными, без приведения их к типу byte;
- Метод write(byte b[]) предназначен для записи в выходной поток данных целочисленного массива байтов, переданных в качестве аргумента;
- Метод write(byte b[], int off, int len) необходим для записи в поток данных, только определенной части массива-аргумента, длиной len байтов, начиная с элемента с порядковым номером off,т.е. С элемента b[off];
- Метод flush() предназначен для очистки выходного буфера, с последующим завершением операции вывода данных;
- Метод close()предназначен для закрытия выходного потока. После исполнения этого метода, поток закрывается. Любые обращения к его данным, попытки записи данных в поток, приводят к генерированию ошибки IOException.

Файловые потоки

Наиболее важный вид потоков, файловый, представлен несколькими классами. Класс FileInputStream предназначен для считывания данных из файлов.

Пример использования одного и того же файла несколькими объектами класса FileInputStream.

```
InputStream input0 = new FileInputStream("/readme.txt");
File name = new File("/readme.txt"):
InputStream input1 = new FileInputStream(name);
```

При исполнении кода, происходит создание объекта класса FileInputStream, с одновременным открытием файла в режиме чтения. В классе FileInputStream, который является наследником абстрактного класса InputStream, для реализации задачи, замещены шесть методов. Использование методов mark и reset к объекту класса FileInputStream приводят к генерации исключения IOException.

Пример практического применения существующих методов.

```
import java.util.*;
import java.io.*;
class FIS {
static void public main(String arg[]) throws Exception {
InputStream fio1 = new FileInputStream("/data/readme.txt");
// Создание объекта файлового потока
int iosize;
size = fio1.available(); // получения количества доступных байтов в потоке
System.out.println("Available Bytes: " + size); //вывод количества доступных
байт
System.out.println("1st  25% of the file: read()"); //вывод на экран 25% потока
// считывание побайтно в цикле из потока fio1. Приведение каждого байта к
типу
// char и вывод побайтно на экран
for (int cnt=0; cnt < size/4; cnt++) {System.out.print((char) fio1.read()); }
System.out.println("Available: " + fio1.available());
//вывод количества доступных байт, после вывода порции информации
System.out.println("Reading the next 1/8: read(b[])");//вывод на экран 12,5%
потока
byte arr[] = new byte[size/8];
//создание массива типа байт, размером равным 1/8 объема первоначального
потока.
// использование условного оператора для проверки соответствия считанных
// данных, размера на соответствие 1/8 исходного размера потока
if (fio1.read(arr) != arr.length) {System.err.println("Something wrong");}
// создание объекта tempstring строкового типа, содержащего временную
строку
// длиной arr.length
String tempstring = new String(arr, 0, 0, arr.length);
System.out.  println(tempstring); //вывод строки на экран
System.out.println("Available: " + fio1.available()); //вывод количества
доступных байт
System.out.println("Skip 1/4: skip()"); //Сообщение о пропуске следующих
25% потока
```

```
        fio1.skip(size/4); //пропуск ¼ данных
        System.out.println( "Available: " + fio1.available()); //вывод количества оставшихся байт
        System.out.println("Read len=1/8 to the end of array");
        //Считывание в массив последних данных файла размером 1/8 исходного размера потока.
        // Использование условного оператора для проверки соответствия считанных
        // данных, размера на соответствие 1/8 исходного размера потока.
        if (fio1.read(arr, arr.length-size/8, size/8) != size/8) {
           System.err.println("Something wrong");}
        System.out.println("Available: " + fio1.available()); //вывод количества оставшихся байт
        fio1.close(); //закрытие потока, объект не связан с файлом, файл закрывается
        }}
```

FileOutputStream

Для вывода данных в файл существует класс FileOutputStream. В классе FileOutputStream, также присутствуют два конструктора, аналогичных FileInputStream. Ограничений по созданию объектов данного класса нет и можно создавать объекты, даже если файл не существует. В Java при создании нового объекта типа FileOutputStream происходит создание самого файла, а затем его открытие на запись для вывода, сохранения данных.

Пример практического использования методов класса FileOutputStream. Данные считываются с клавиатуры из потока System.in, до тех пор, пока не будет заполнен 12байтовый буфер, затем данные распределяются по 3м файлам. В первый пишутся все четные байты, во второй все данные, в третий только середина потока, ее 50%.

```
import java.io.*; // считывание всех классов библиотеки java.io
class FIS{
static public byte getlnput()[] throws Exception {
byte buf[] = new byte[12];
for (int cnt=0; cnt<12; cnt++) buf[cnt] = (byte) System.in.read();
return buf;
}
static public void main(String arg[]) throws Exception {
byte buf[] = getlnput();
OutputStream fi0 = new FileOutputStream("1st file.txt");
OutputStream fi1 = new FileOutputStream("2nd file.txt");
OutputStream fi2 = new FileOutputStream("3rd file.txt");
for (int cnt=0; cnt < 12; cnt += 2) fi0.write(buf[cnt]);
fi0.close();
fi1.write(buf);
fi1.close();
fi2.write(buf, 12/4, 12/2);
fi2.close();
} }
```

Есть существенный недостаток Java, который выражается отсутствием способа, возможности открыть файл для дозаписи FileOutputStream. При открытии файла с помощью штатного конструктора FileOutputStream, содержимое файла стирается.

ByteArraylnputStream

Универсальный класс ByteArrayInputStream предназначен для реализации входного потока. Этот класс для тех случаев, когда в качестве источника применяется массив типа byte. Как у предыдущих классов этот класс имеет два конструктора. Разница только в том, что первым обязательным параметром является байтовый массив.

StringBufferlnputStream

Особый класс StringBufferInputStream, практически, идентичен классу ByteArrayInputStream. Разница заключается в том, что внутренний буфер объекта типа StringBufferInputStream создан не как байтовый массив, а как String. Особенность класса в том, что в нем есть только один конструктор StringBufferInputStream(String s) и нет симметричного класса ввода StringBufferedOutputStream.

Вывод

Реализованная в Java объектно-ориентированная концепция, позволяет программам, написанным на основании существующих классов, работать, и в будущем, на новых классах ввода/вывода с другой внутренней реализацией. Эта модель позволяет легко переходить от ориентированных на файлы наборов потоков, к работе с сетевыми потоками.

Существующие потоки Java требуют внимательного изучения и применения на практике методов, так чтобы в прикладных задачах грамотно использовать разные классы потоков ввода/вывода, реализацию сложных, комплексных операций. Для успешной сдачи на отлично блока вопросов, связанных с потоками ввода/вывода, важно иметь ясное представление о существующих классах, четкое понимание синтаксиса и функционала, существующей иерархии классов, начиная с абстрактных InputStream и OutputStream.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В виде чего представляют абстракцию потока?
2. Опишите иерархию классов ввода\вывода.
3. Из какого класса наследуются все классы работающие с вводом, с выводом?
4. Виды источников вывода и ввода?
5. Система хранения каталогов на Java?
6. Какие методы содержит класс InputStream?
7. Какие методы содержит класс OutputStream?
8. Для чего служит BufferedOutputStream?

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Задание 1.

Создать файл с информацией о себе. Вывести данные об этом файле. Вывести информацию из файла.

Задание 2.

Ввести с клавиатуры в файл f 10 целых чисел. Затем открыть файл для чтения, считать числа, вывести их на экран и определить среднее значение.

Задание 3.

Создать текстовый файл f, компонентами которого являются целые случайные числа. Записать в файл g все четные числа файла из f, а в файл h – все нечетные. Порядок следования чисел сохраняется.

Задание 4.

Ввести с клавиатуры в файл 15 вещественных значений температуры воздуха. Затем создать программу, считывающую из файла значения и выводящую на экран среднюю температуру.

Задание 5

Создать текстовый файл f и записать в него 10 целых чисел. Затем считать из файла числа и вывести на экран количество положительных значений.

ДОМАШНЕЕ ЗАДАНИЕ

Индивидуальное задание

ЛИТЕРАТУРА

У. Савитч, язык Java. Курс программирования, Вильямс, 2015

Преподаватель А.С.Кибисова

Рассмотрено на заседании цикловой комиссии программного обеспечения информационных технологий
Протокол № ____ от «___» _____ 2021
Председатель ЦК _____В.Ю.Михалевич