Ввод-вывод в Java

Пакет java.io

Пакет java.io

- Концепция работы в пакете java.io включает две составляющие:
  - Работа с файлами и каталогами с помощью объектов File.
  - Работа с потоками ввода-вывода.

Создание объекта типа File осуществляется с помощью конструкторов, имеющих следующие варианты:

File(”*Имя каталога*”)

File(”*Имя файла*”)

File(”*Имя папки*”,”*Имя каталога*”).

Для платформы Windows® вместо символа ”\” в строках, соответствующих путям, должна использоваться последовательность ”\\”.

Важнейшие методы

класса File

Пример работы с файловыми объектами:

```
File f1=new File(".."); // "." , "/" , "C:/../"
System.out.println
                    ("AbsolutePath"+f1.getAbsolutePath());
System.out.println("exists(): "+f1.exists());
System.out.println("canRead(): "+f1.canRead());
System.out.println("canWrite(): "+f1.canWrite());
```

Отображение списка директории

```
import java.io.*;
import java.util.*;

public class DirList {
        public static void main(String[ ] args) {
                File path = new File(".");
                String[ ] list;
                if(args.length == 0) list = path.list();
                else list =
                path.list(new DirFilter(args[0]));
                for(int i = 0; i < list.length; i++)
                        System.out.println(list[i]);
} }
```

Класс DirFilter

```
class DirFilter implements FilenameFilter {
Private Pattern ptr;
DirFilter(String afn) {
            ptr = Pattern.compile( afn);
           }
public boolean accept(File dir, String name) {
   return ptr.matcher(name).matches();
}
}
```

Работа с потоками ввода-вывода

Поток представляет накапливающуюся последовательность данных, поступающих из какого-то источника.

Порция данных может быть считана из потока, при этом она из потока изымается. В потоке действует принцип очереди – “первым вошёл, первым вышел”.

В качестве источника данных потока может быть использован как стационарный источник данных (файл, массив, строка), так и динамический – другой поток. При этом в ряде случаев выход одного потока может служить входом другого.

Буферизуемые потоки имеют хранящийся в памяти промежуточный буфер, из которого считываются выходные данные потока. Наличие такого буфера позволяет повысить производительность операций ввода-вывода, а также осуществлять дополнительные операции – устанавливать метки (маркеры) для какого-либо элемента, находящегося в буфере потока и даже делать возврат считанных элементов в поток (в пределах буфера).

Абстрактный класс InputStream

Методы класса InputStream:

int available()	Текущее количество байт, доступных для чтения из потока.
int read()	Читает один байт из потока и возвращает его значение в виде целого, лежащего в диапазоне от 0 до 255. При достижении конца потока возвращает -1
int read(byte[] b) int read(byte[] b, int offset, int count)	Пытается прочесть b.length байт из потока в массив b. Возвращает число реально прочитанных байт. После достижения конца потока следующие считывания возвращают -1.

Абстрактный класс InputStream

long skip(long count)	Попытка пропускать count байт из потока. Возвращается реально пропущенное число байт. Если это значение ≤0, пропуска байт не было.
boolean markSupported()	Возвращает true в случае, когда поток поддерживает операции mark и reset, иначе – false.

Абстрактный класс InputStream

void mark(int limit)	Ставит метку в текущей позиции начала потока. Используется для последующего вызова метода reset, с помощью которого считанные после установки метки данные возвращаются обратно в поток. Эту операцию поддерживают не все потоки.

Абстрактный класс InputStream

void reset()	Восстанавливает предшествующее состояние данных в начале потока, возвращая указатель начала потока на помеченный до того меткой элемент. То есть считанные после установки метки данные возвращаются обратно в поток. Попытка вызова при отсутствии метки или выходе её за пределы лимита приводит к возбуждению исключения IOException. Эту операцию поддерживают не все потоки

Абстрактный класс InputStream

void close()	Закрытие потока. Последующие попытки чтения из этого потока приводят к возбуждению исключения IOException.

Все методы класса InputStream, кроме markSupported и mark, возбуждают исключение IOException.

Разновидности входных потоков InputStream

- ByteArrayInputStream поток - массив байтов
- StringBufferInputStream поток - строка
- FileInputStream поток - файл
- PipedInputStream поток - PipedOutputStream
- SequenceInputStream объединение потоков
- FilterInputStream абстрактный класс для реализации надстроек

Абстрактный класс OutputStream

Методы класса OutputStream:

void write(int b)	Записывает один байт в поток. Благодаря использованию типа int можно использовать в качестве параметра целочисленное выражение без приведения его к типу byte.

Абстрактный класс OutputStream

void write(byte[] b) void write(byte[] b, int offset, int count)	Записывает в поток массив байт. Если заданы параметры offset и count, записывается не весь массив, а count байт начиная с индекса offset.
void flush()	Форсирует вывод данных из выходного буфера и его очистку.
void close()	Закрытие потока. Последующие попытки записи в этот поток приводят к возбуждению исключения IOException.

Абстрактный класс OutputStream

Все методы этого класса возбуждают IOException в случае ошибки записи.

Разновидности выходных потоков OutputStream

- ByteArrayOutputStream поток - массив байтов
- FileOutputStream поток - файл
- PipedOutputStream поток - PipedOutputStream
- FilterOutputStream абстрактный класс для реализации надстроек

Пример

```
import java.io.*;
public class BaseFileInput {
 public static void main (String args[]){
  if (args.length!=1){
 System.out.println("input file name in parameters");return; }
InputStream myFile;
try {  myFile = new FileInputStream(args[0]);
  while (myFile.available()>0) {
    System.out.print((char)myFile.read());  }
 myFile.close();
} catch (Exception e) {
  System.err.println(e.getMessage());
 }}}
```

Пример

```
OutputStream myFile;
 int myKey;
 try{
    myFile = new FileOutputStream(“…”);
   System.out.println("for exit press Ctrl+z");
   while ((myKey=System.in.read())!=-1){
                        myFile.write(myKey);}
  myFile.flush();
  myFile.close();
} catch(Exception e){
  System.out.println(e.getMessage());
 }
```

Надстройки для входных потоков FilterInputStream

- DataInputStream
- BufferedInputStream
- LineNumberInputStream
- PushbackInputStream

Надстройки для выходных потоков FilterOutputStream

- DataOutputStream
- BufferedOutputStream
- PrintStream

Классы Reader и Writer

InputStreamReader	OutputStreamWriter
FileReader	FileWriter
StringReader	StringWriter
CharArrayReader	CharArrayWriter
PipedReader	PipedWriter
FilterReader	FilterWriter
BufferedReader	BufferedWriter
LineNumberReader	PrintWriter
PushBackReader	

Дополнительные классы

RandomAccessFile

StreamTokenizer

Новый ввод/вывод

- Пакет java.nio.*

начиная с версии JDK 1.4

- Основные структуры
  - каналы (channels) и
  - буферы (buffers)

  близкие к средствам операционной системы

Сериализация объектов

- Объект должен реализовывать интерфейс Serializable
- Классы ObjectInputStream и ObjectOutputStream
- Методы readObject() и writeObject()