Работа с файлами. Основные классы. (работа через потоки)

import java.io.\*;

**Об исключениях:**

! Используется семейство исключений IOException

! Любой метод работы с файлами может выбросить исключение и требует, чтобы оно было обработано, например, закрыть файл при аварии:

try{

//некоторые подготовительные действия, указатели на потоки, файлы

try{

//открытие/закрытие файла на чтение/запись, чтение и запись

//ошибки чтения/записи желательно обработать без исп.исключений

например, если файл не удалось открыть, ни читать\не писать в него.

}catch(IOException){//действия при ошибке чтения, непредусмотренной

}finally{например, мы должны закрыть файл в любом случае}

}catch(IOException e){обработка исключений} // здесь файл закрывать нельзя – при ошибке будет исключение, которое обработать уже некому.

! можно поймать исключение и заменить его своим, более конкретным.

**Класс File:**

File file = new File("res/filename"); //файл или каталог

getAbsoluteFile(); //полный путь + имя файла

mkdirs() - создание каталога с указанным в конструкторе именем

String[] list() - список файлов и каталогов в текущем каталоге

**<<<БАЙТОВЫЕ ПОТОКИ>>>**

(abstract)**InputStream**: - входной поток байтов

int available() - сколько байт еще осталось в потоке

int read() – читаем 1 байт

int read(b) – чтение в byte[] b до конца массива или файла

int read(b,i,n) – заполнение части массива, чтение максимум n байт

skip(long n) – пропустить (отбросить) n байт входного потока.

close() – (внутри finally, можно ещё внутри try) закрыть поток

(i1) **FileInputStream** – байтовый поток чтения из нетекстового файла

FileInputStream in=null; - создание ссылки вне try и её зануление

in=new FileInputStream(file) – (внутри try) открыть файл, вернёт указатель

byte[] in.readAllBytes()–чтение всего потока ввода, сохранение в виде массива

byte[] in.readNBytes(n)-чтение n байт из потока, сохранение в виде массива.

(i2) **FilterInputStream** - содержит некоторый другой входной поток, который он использует в качестве основного источника данных, возможно, преобразуя данные по пути. Подклассы FilterInputStream могут дополнительно переопределять некоторые из этих методов.

(i2.1) **DataInputStream**

new DataInputStream(InputStream) - конструктор

readBooolean(), readByte(int), readChar(int), readShort(), readInt(), readDouble(), readFloat(), readLong(), readUTF() Данный класс с обычными строками работает некорректно, лучше использовать другие классы:

DataInputStream d = new DataInputStream(in);

BufferedReader d=new BufferedReader(new InputStreamReader(in));

(i2.2) **BufferedInputStream** – существующий поток+буфер, разом заполняет внутренний буфер данными с утройства, далее данные читаются с буфера, после опустения буфера данные снова читаются в буфер с устройства.

BufferedInputStream(InputStream), BufferedInputStream(InputStream, bufSize)

(i3) ByteArrayInputStream

(i4) ObjectInputStream

(abstract)**OutputStream**: - выходной поток байтов

flush() - сброс потока, запись данных из буфера, очистка буфера

out.write(a) – запись в поток/файл, где а это int или ссылка на массив byte[]

out.white(a,i,n) – запись среза массива byte[] a, i – индекс, n-число байт

(o1) **FileOutputStream** - байтовый поток записи в нетекстовый файл.

FileOutputStream out=null;

out=new FileOutputStream(file);

write(s1.getBytes(), 0, s1.length());//пишем s1, строку String

(o2) FilterOutputStream - суперкласс всех классов, фильтрующих выходные потоки. Эти потоки располагаются поверх уже существующего выходного потока (базового выходного потока), который он использует в качестве основного приемника данных.

(o2.1) DataOutputStream

new DataOutputStream(OutputStream) - конструктор

writeBooolean(), writeByte(int), writeChar(int),writeShort(), writeInt(), writeDouble(), writeFloat(), writeLong(), writeBytes(String s), writeUTF(String s)

(i2.2) BufferedOutputStream - существующий поток+буфер, постепенно осуществляет запись данных во внутренний буфер, после заполнения буфера производится запись всех данных буфера на устройство.

(o3) ByteArrayOutputStream

(o4) ObjectOutputStream

in=new DataOutputStream(FileInputStream(file)); - создание О

Класс DataOutputStream позволяет писать элементарные данные в поток через интерфейс DataOutput, который определяет методы, преобразующие элементарные значения в форму последовательности байтов.

writeBoolean(boolean v) : записывает в поток булевое однобайтовое значение

writeDouble(double v): записывает в поток 8-байтовое значение double

writeInt(int v): записывает в поток целочисленное значение int

writeUTF(String str): записывает в поток строку в кодировке UTF-8

+byte,short,char,float,long

Класс DataInputStream действует противоположным образом.

В методах выше меняем write на read,например, readShort() возвращает short

int skipBytes(int n): пропускает при чтении из потока n байтов

(2.2) BufferedOutputStream

**<<<СИМВОЛЬНЫЕ ПОТОКИ>>>**

(abstract)Reader:

InputSteamReader:

FileReader:

read() - читаем 2х байтовый символ

FileWriter:

write() - записываем 2х байтовый символ

**<<<СТАНДАРТНЫЕ ПОТОКИ>>>**

System.in — стандартный ввод (клавиатура)

System.out — стандартный вывод (консоль)

System.err — стандартный вывод сообщений об ошибках (консоль)

InputSteamReader isr;

isr=new InputSteamReader(System.in);//подключаем станд. ввод к потоку

isr.read() - чтение символа из потока (после нажатия enter)