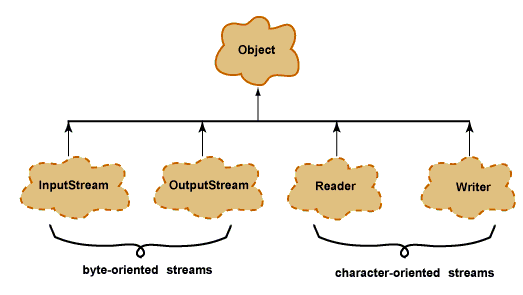
***Какие существуют виды потоков ввода/вывода?***

Разделяют два вида потоков ввода/вывода: байтовые и символьные.



Java имеет в своём составе множество классов, связанных с вводом/выводом данных.

***Класс File***

В отличие от большинства классов ввода/вывода, класс File работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов.

В отличие от большинства классов ввода/вывода, класс File работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов.

Класс java.io.File может представлять имя определённого файла, а также имена группы файлов, находящихся в каталоге. Если класс представляет каталог, то его метод list() возвращает массив строк с именами всех файлов.

Для создания объектов класса File можно использовать один из следующих конструкторов.

File(File dir, String name) - указывается объекта класса File (каталог) и имя файла

File(String path) - указывается путь к файлу без указания имени файла

File(String dirPath, Sring name) - указывается путь к файлу и имя файла

File(URI uri) - указывается объекта URI, описывающий файл

**Методы класса File**

Класс File может использоваться для создания каталога или дерева каталогов. Также можно узнать свойства файлов (размер, дату последнего изменения, режим чтения/записи), определить к какому типу (файл или каталог) относится объект File, удалить файл. У класса очень много методов, перечислим некоторые.

getAbsolutePath() - абсолютный путь файла, начиная с корня системы. В Android корневым элементом является символ слеша (/)

canRead() - доступно для чтения

canWrite() - доступно для записи

exists() - файл существует или нет

getName() - возвращает имя файла

getParent() - возвращает имя родительского каталога

getPath() - путь

lastModified() - дата последнего изменения

isFile() - объект является файлом, а не каталогом

isDirectory - объект является каталогом

isAbsolute() - возвращает true, если файл имеет абсолютный путь

renameTo(File newPath) - переименовывает файл. В параметре указывается имя нового имени файла. Если переименование прошло неудачно, то возвращается false

delete() - удаляет файл. Также можно удалить пустой каталог

**Каталог**

Каталог - это объект класса File, который содержит список других файлов и каталогов. После создания объекта класса File, являющего каталогом, его метод isDirectory() вернёт значение true. И тогда вы можете вызывать метод list().

Для создания каталога можно использовать метод mkdir(), который вернёт true в успешном случае. Если указанный путь уже существует или каталог нельзя создать из-за отсутствия полного пути к нему, то вернётся false.

Метод mkdirs() создаёт сам каталог, так и всех его родителей.

**Список каталогов**

Если вы хотите получить содержимое каталога, то можно вызвать метод list() без аргументов. Вам вернётся полный список (массив) имён файлов и каталогов, содержащихся в данном каталоге. Есть ещё похожий метод listFiles(), который возвращает массив файлов (объектов, а не их имён), о котором поговорим отдельно.

String dirPath = "/";

File file = new File(dirPath);

File[] files = file.listFiles();

Если вам нужен только список файлов с расширением .java, то можно использовать специальный фильтр-класс, который описывает критерии отбора объектов File с помощью интерфейса FilenameFilter.

String[] list(FilenameFilter filter)

Интерфейс FilenameFilter определяет единственный метод accept(), вызываемый по одному разу с каждым файлом в списке.

boolean accept(File dir, String filename)

Метод возвращает true для файлов каталога, указанного в dir, которые должны быть включены в список, и false - для файлов, которые следует исключить.

**Метод listFiles()**

Метод имеет несколько перегруженных версий. Он возвращает список файлов в виде массива объектов класса File. Одна из версий метода также использует интерфейс FilenameFilter. Другая версия использует интерфейс FileFilter и возвращает те файлы, путевые имена которых соответствует интерфейсу.

**Поток**

При работе с данными ввода/вывода вам будет часто попадаться термин Поток (Stream). Поток - это абстрактное значение источника или приёмника данных, которые способны обрабатывать информацию. Вы в реальности не видите, как действительно идёт обработка данных в устройствах ввода/вывода, так как это сложно и вам это не нужно. Это как с телевизором - вы не знаете, как сигнал из кабеля превращается в картинку на экране, но вполне можете переключаться между каналами через пульт.

Есть два типа потоков: байтовые и символьные. В некоторых ситуациях символьные потоки более эффективны, чем байтовые.

За ввод и вывод отвечают разные классы Java. Классы, производные от базовых классов InputStream или Reader, имеют методы с именами read() для чтения отдельных байтов или массива байтов (отвечают за ввод данных). Классы, производные от классов OutputStream или Write, имеют методы с именами write() для записи одиночных байтов или массива байтов (отвечают за вывод данных).

**Класс InputStream**

Базовый класс InputStream представляет классы, которые получают данные из различных источников:

* массив байтов
* строка (String)
* файл
* канал (pipe): данные помещаются с одного конца и извлекаются с другого
* последовательность различных потоков, которые можно объединить в одном потоке
* другие источники (например, подключение к интернету)

Для работы с указанными источниками используются подклассы базового класса InputStream:

BufferedInputStream

Буферизированный входной поток

ByteArrayInputStream

Позволяет использовать буфер в памяти (массив байтов) в качестве источника данных для входного потока.

DataInputStream

Входной поток, включающий методы для чтения стандартных типов данных Java

FileInputStream

Для чтения информации из файла

FilterInputStream

Абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства.

InputStream

Абстрактный класс, описывающий поток ввода

ObjectInputStream

Входной поток для объектов

StringBufferInputStream

Превращает строку (String) во входной поток данных InputStream

PipedInputStream

Реализует понятие входного канала.

PushbackInputStream

Входной поток, поддерживающий однобайтовый возврат во входной поток

SequenceInputStream

Сливает два или более потока InputStream в единый поток

**Методы класса:**

int available() - возвращает количество байтов ввода, доступные в данный момент для чтения

close() - закрывает источник ввода. Следующие попытки чтения передадут исключение IOException

void mark(int readlimit) - помещает метку в текущую точку входного потока, которая остаётся корректной до тех пор, пока не будет прочитано readlimint байт

boolean markSupported() - возвращает true, если методы mark() и reset() поддерживаются потоком

int read() - возвращает целочисленное представление следующего доступного байта в потоке. При достижении конца файла возвращается значение -1

int read(byte[] buffer) - пытается читать байты в буфер, возвращая количество прочитанных байтов. По достижении конца файла возвращает значение -1

int read(byte[] buffer, int byteOffset, int byteCount) - пытается читать до byteCount байт в buffer, начиная с смещения byteOffset. По достижении конца файла возвращает -1

reset() - сбрасывает входной указатель в ранее установленную метку

long skip(long byteCount) - пропускает byteCount байт ввода, возвращая количество проигнорированных байтов

**Как преобразовать InputStream в строку**

На stackoverflow дали [развёрнутый ответ из 11 вариантов](http://stackoverflow.com/questions/309424/read-convert-an-inputstream-to-a-string). Есть [вариант на русском](https://habrahabr.ru/company/luxoft/blog/278233/).

1. Using IOUtils.toString (Apache Utils):

String result = IOUtils.toString(inputStream, StandardCharsets.UTF\_8);

1. Using CharStreams (guava)

String result = CharStreams.toString(new InputStreamReader(

inputStream, Charsets.UTF\_8));

1. Using Scanner (JDK)  
   символ «\А» является символом начала текста, таким образом вызов next() вернет сразу всю строку.

Scanner s = new Scanner(inputStream).useDelimiter("\\A");

String result = s.hasNext() ? s.next() : "";

1. Using Stream Api (Java 8). Warning: This solution convert different linebreaks (like \r\n) to \n.

String result = new BufferedReader(new InputStreamReader(inputStream))

.lines().collect(Collectors.joining("\n"));

1. Using parallel Stream Api (Java 8). Warning: This solution convert different linebreaks (like \r\n) to \n.

String result = new BufferedReader(new InputStreamReader(inputStream)).lines()

.parallel().collect(Collectors.joining("\n"));

1. Using InputStreamReader and StringBuilder (JDK)

final int bufferSize = 1024;

final char[] buffer = new char[bufferSize];

final StringBuilder out = new StringBuilder();

Reader in = new InputStreamReader(inputStream, "UTF-8");

for (; ; ) {

int rsz = in.read(buffer, 0, buffer.length);

if (rsz < 0)

break;

out.append(buffer, 0, rsz);

}

return out.toString();

1. Using StringWriter and IOUtils.copy (Apache Commons)

StringWriter writer = new StringWriter();

IOUtils.copy(inputStream, writer, "UTF-8");

return writer.toString();

1. Using ByteArrayOutputStream and inputStream.read (JDK)

ByteArrayOutputStream result = new ByteArrayOutputStream();

byte[] buffer = new byte[1024];

int length;

while ((length = inputStream.read(buffer)) != -1) {

result.write(buffer, 0, length);

}

return result.toString("UTF-8");

1. Using BufferedReader (JDK). Warning: This solution convert different linebreaks (like \n\r) to line.separator system property (for example, in Windows to "\r\n").

String newLine = System.getProperty("line.separator");

BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));

StringBuilder result = new StringBuilder();

String line; boolean flag = false;

while ((line = reader.readLine()) != null) {

result.append(flag? newLine: "").append(line);

flag = true;

}

return result.toString();

1. Using BufferedInputStream and ByteArrayOutputStream (JDK)

BufferedInputStream bis = new BufferedInputStream(inputStream);

ByteArrayOutputStream buf = new ByteArrayOutputStream();

int result = bis.read();

while(result != -1) {

buf.write((byte) result);

result = bis.read();

}

return buf.toString();

1. Using inputStream.read() and StringBuilder (JDK). Warning: This soulition has problem with Unicode, for example with Russian text (work correctly only with non-Unicode text)

int ch;

StringBuilder sb = new StringBuilder();

while((ch = inputStream.read()) != -1)

sb.append((char)ch);

reset();

return sb.toString();

Warning:  
Solutions 4, 5 and 9 convert different linebreaks to one.  
Soulution 11 can't work correclty with Unicode text

**BufferedInputStream**

Буферизация ввода-вывода является удобным способом оптимизации производительности, позволяя заключить в оболочку любой поток класса InputStream.

У класса есть конструктор, где размер буфера устанавливается по умолчанию. Также можно использовать конструктор, где размер буфера устанавливается вручную. Рекомендуется использовать размеры буфера, кратные размеру страницы памяти, дисковому блоку и т.п. и может зависеть от принимающей операционной системы, объёма доступной памяти и конфигурации машины.

**ByteArrayInputStream**

Класс ByteArrayInputStream использует байтовый массив в качестве источника данных. У данного класса можно не вызывать метод close().

**DataInputStream - Форматированное чтение из памяти**

Для чтения байтовых данных (не строк) применяется класс DataInputStream. В этом случае необходимо использовать классы из группы InputStream.

Для преобразования строки в массив байтов, пригодный для помещения в поток ByteArrayInputStream, в классе String предусмотрен метод getBytes(). Полученный ByteArrayInputStream представляет собой поток InputStream, подходящий для передачи DataInputStream.

При побайтовом чтении символов из форматированного потока DataInputStream методом readByte() любое полученное значение будет считаться действительным, поэтому возвращаемое значение неприменимо для идентификации конца потока. Вместо этого можно использовать метод available(), который сообщает, сколько еще осталось символов.

Класс DataInputStream позволяет читать элементарные данные из потока через интерфейс DataInput, который определяет методы, преобразующие элементарные значения в форму последовательности байтов. Такие потоки облегчают сохранение в файле двоичных данных.

**Конструктор:**

DataInputStream(InputStream stream)

**Методы:**

readDouble()

readBoolean()

readInt()

**FileInputStream**

Класс FileInputStream создаёт объект класса InputStream, который можно использовать для чтения байтов из файла.

Конструкторы

FileInputStream (File file) - указывается объекта типа File

FileInputStream (FileDescriptor fd)

FileInputStream (String path) - указывается полное имя файла

При создании объект открывается для чтения. Класс переопределяет методы класса InputStream, кроме методов mark() и reset().

Для чтения байтов входного потока из файла используется конструкция:

File file = ...

InputStream in = null;

try {

in = new BufferedInputStream(new FileInputStream(file));

...

finally {

if (in != null) {

in.close();

}

}

}

**PushbackInputStream**

Разновидность буферизации, обеспечивающая чтение байта с последующим его возвратом в поток. Класс PushbackInputStream представляет механизм "заглянуть" во входной поток и увидеть, что оттуда поступит в следующий раз, не извлекая информации.

У класса есть дополнительный метод unread().

**SequenceInputStream**

Класс SequenceInputStream позволяет соединять вместе несколько экземпляров класса InputStream. Конструктор принимает в качестве аргумента либо пару объектов класса InputStream, либо интерфейс Enumeration.

Во время работы класс выполняет запросы на чтение из первого объекта класса InputStream и до конца, а затем переключается на второй. При использовании интерфейса работа продолжится по всем объектам класса InputStream. По достижении конца каждого файла, связанный с ним поток закрывается. Закрытие потока, созданного объектом класса SequenceInputStream, приводит к закрытию всех открытых потоков.

Класс OutputStream

Класс OutputStream - это абстрактный класс, определяющий потоковый байтовый вывод.

В этой категории находятся классы, определяющие, куда направляются ваши данные: в массив байтов (но не напрямую в String; предполагается что вы сможете создать их из массива байтов), в файл или канал.

BufferedOutputStream

Буферизированный выходной поток

ByteArrayOutputStream

Создает буфер в памяти. Все данные, посылаемые в этот поток, размещаются в созданном буфере

DataOutputStream

Выходной поток, включающий методы для записи стандартных типов данных Java

FileOutputStream

Отправка данных в файл на диске. Реализация класса OutputStream

ObjectOutputStream

Выходной поток для объектов

PipedOutputStream

Реализует понятие выходного канала.

FilterOutputStream

Абстрактный класс, предоставляющий интерфейс для классов-надстроек, которые добавляют к существующим потокам полезные свойства.

Методы класса:

int close() - закрывает выходной поток. Следующие попытки записи передадут исключение IOException

void flush() - финализирует выходное состояние, очищая все буферы вывода

abstract void write (int oneByte) - записывает единственный байт в выходной поток

void write (byte[] buffer) - записывает полный массив байтов в выходной поток

void write (byte[] buffer, int offset, int count) - записывает диапазон из count байт из массива, начиная с смещения offset

**BufferedOutputStream**

Класс BufferedOutputStream не сильно отличается от класса OutputStream, за исключением дополнительного метода flush(), используемого для обеспечения записи данных в буферизируемый поток. Буферы вывода нужно для повышения производительности.

**ByteArrayOutputStream**

Класс ByteArrayOutputStream использует байтовый массив в выходном потоке. Метод close() можно не вызывать.

**DataOutputStream**

Класс DataOutputStream позволяет писать элементарные данные в поток через интерфейс DataOutput, который определяет методы, преобразующие элементарные значения в форму последовательности байтов. Такие потоки облегчают сохранение в файле двоичных данных.

Класс DataOutputStream расширяет класс FilterOutputStream, который в свою очередь, расширяет класс OutputStream.

Методы интерфейса DataOutput:

writeDouble(double value)

writeBoolean(boolean value)

writeInt(int value)

**FileOutputStream**

Класс FileOutputStream создаёт объект класса OutputStream, который можно использовать для записи байтов в файл. Создание нового объекта не зависит от того, существует ли заданный файл, так как он создаёт его перед открытием. В случае попытки открытия файла, доступного только для чтения, будет передано исключение.

**Классы символьных потоков**

Символьные потоки имеют два основных абстрактных класса Reader и Writer, управляющие потоками символов Unicode.

**Reader**

Методы класса Reader:

abstract void close() - закрывает входной поток. Последующие попытки чтения передадут исключение IOException

void mark(int readLimit) - помещает метку в текущую позицию во входном потоке

boolean markSupported() - возвращает true, если поток поддерживает методы mark() и reset()

int read() - возвращает целочисленное представление следующего доступного символа вызывающего входного потока. При достижении конца файла возвращает значение -1. Есть и другие перегруженные версии метода

boolean ready() - возвращает значение true, если следующий запрос не будет ожидать.

void reset() - сбрасывает указатель ввода в ранее установленную позицию метки

logn skip(long charCount) - пропускает указанное число символов ввода, возвращая количество действительно пропущенных символов

BufferedReader

Буферизированный входной символьный поток

CharArrayReader

Входной поток, который читает из символьного массива

FileReader

Входной поток, читающий файл

FilterReader

Фильтрующий читатель

InputStreamReader

Входной поток, транслирующий байты в символы

LineNumberReader

Входной поток, подсчитывающий строки

PipedReader

Входной канал

PushbackReader

Входной поток, позволяющий возвращать символы обратно в поток

Reader

Абстрактный класс, описывающий символьный ввод

StringReader

Входной поток, читающий из строки

Класс BufferedReader

Класс BufferedReader увеличивает производительность за счёт буферизации ввода.

Класс CharArrayReader

Класс CharArrayReader использует символьный массив в качестве источника.

Класс FileReader

Класс FileReader, производный от класса Reader, можно использовать для чтения содержимого файла. В конструкторе класса нужно указать либо путь к файлу, либо объект типа File.

**Writer**

Класс Writer - абстрактный класс, определяющий символьный потоковый вывод. В случае ошибок все методы класса передают исключение IOException.

Методы класса:

Writer append(char c) - добавляет символ в конец вызывающего выходного потока. Возвращает ссылку на вызывающий поток

Writer append(CharSequence csq) - добавляет символы в конец вызывающего выходного потока. Возвращает ссылку на вызывающий поток

Writer append(CharSequence csq, int start, int end) - добавляет диапазон символов в конец вызывающего выходного потока. Возвращает ссылку на вызывающий поток

abstract void close() - закрывает вызывающий поток

abstract void flush() - финализирует выходное состояние так, что все буферы очищаются

void write(int oneChar) - записывает единственный символ в вызывающий выходной поток. Есть и другие перегруженные версии метода

BufferedWriter

Буферизированный выходной символьный поток

CharArrayWriter

Выходной поток, который пишет в символьный массив

FileWriter

Выходной поток, пишущий в файл

FilterWriter

Фильтрующий писатель

OutputStreamWriter

Выходной поток, транслирующий байты в символы

PipedWriter

Выходной канал

PrintWriter

Выходной поток, включающий методы print() и println()

StringWriter

Выходной поток, пишущий в строку

Writer

Абстрактный класс, описывающий символьный вывод

**Класс BufferedWriter**

Класс BufferedWriter - это класс, производный от класса Writer,который буферизует вывод. С его помощью можно повысить производительность за счёт снижения количества операций физической записи в выходное устройство.

Класс CharArrayWriter

Класс CharArrayWriter использует массив для выходного потока.

Класс FileWriter

Класс FileWriter создаёт объект класса, производного от класса Writer, который вы можете применять для записи файла. Есть конструкторы, которые позволяют добавить вывод в конец файла. Создание объекта не зависит от наличия файла, он будет создан в случае необходимости. Если файл существует и он доступен только для чтения, то передаётся исключение IOException.

**Чтение и запись файлов**

Существует множество классов и методов для чтения и записи файлов. Наиболее распространённые из них - классы FileInputStream и FileOutputStream, которые создают байтовые потоки, связанные с файлами. Чтобы открыть файл, нужно создать объект одного из этих файлов, указав имя файла в качестве аргумента конструктора.

FileInputStream(String filename) throws FileNotFoundException

FileOutputStream(String filename) throws FileNotFoundException

В filename нужно указать имя файла, который вы хотите открыть. Если при создании входного потока файл не существует, передаётся исключение FileNotFoundException. Аналогично для выходных потоков, если файл не может быть открыт или создан, также передаётся исключение. Сам класс исключения происходит от класса IOException. Когда выходной файл открыт, любой ранее существовавший файл с тем же именем уничтожается.

После завершения работы с файлом, его необходимо закрыть с помощью метода close() для освобождения системных ресурсов. Незакрытый файл приводит к утечке памяти.

В JDK 7 метод close() определяется интерфейсом AutoCloseable и можно явно не закрывать файл, а использовать новый оператор try-с-ресурсами, что для Android пока не слишком актуально.

Чтобы читать файл, нужно вызвать метод read(). Когда вызывается этот метод, он читает единственный байт из файла и возвращает его как целое число. Когда будет достигнут конец файла, то метод вернёт значение -1. Примеры использования методов есть в различных статьях на сайте.

Иногда используют вариант, когда метод close() помещается в блок finally. При таком подходе все методы, которые получают доступ к файлу, содержатся в пределах блока try, а блок finally используется для закрытия файла. Таким образом, независимо от того, как закончится блок try, файл будет закрыт.

Так как исключение FileNotFoundException является подклассом IOException, то не обязательно обрабатывать два исключения отдельно, а оставить только IOException, если вам не нужно отдельно обрабатывать разные причины неудачного открытия файла. Например, если пользователь вводит вручную имя файла, то более конкретное исключение будет к месту.

Для записи в файл используется метод write().

void write(int value) throws IOException

Метод пишет в файл байт, переданный параметром value. Хотя параметр объявлена как целочисленный, в файл записываются только младшие восемь бит. При ошибке записи передаётся исключение.

В JDK 7 есть способ автоматического управления ресурсами:

try (спецификация\_ресурса) {

// использование ресурса

}

**Буферизированное чтение из файла - BufferedReader**

Чтобы открыть файл для посимвольного чтения, используется класс FileInputReader; имя файла задаётся в виде строки (String) или объекта File. Ускорить процесс чтения помогает буферизация ввода, для этого полученная ссылка передаётся в конструктор класса BufferedReader. Так как в интерфейсе класса имеется метод readLine(), все необходимое для чтения имеется в вашем распоряжении. При достижении конца файла метод readLine() возвращает ссылку null.

**Вывод в файл - FileWriter**

Объект FileWriter записывает данные в файл. При вводе/выводе практически всегда применяется буферизация, поэтому используется BufferedWriter.

Когда данные входного потока исчерпываются, метод readLine() возвращает null. Для потока явно вызывается метод close(); если не вызвать его для всех выходных файловых потоков, в буферах могут остаться данные, и файл получится неполным.

**Сохранение и восстановление данных - PrintWriter**

PrintWriter форматирует данные так, чтобы их мог прочитать человек. Однако для вывода информации, предназначенной для другого потока, следует использовать классы DataOutputStream для записи данных и DataInputStream для чтения данных.

Единственным надежным способом записать в поток DataOutputStream строку так, чтобы ее можно было потом правильно считать потоком DataInputStream, является кодирование UTF-8, реализуемое методами readUTF() и writeUTF(). Эти методы позволяют смешивать строки и другие типы данных, записываемые потоком DataOutputStream, так как вы знаете, что строки будут правильно сохранены в Юникоде и их будет просто воспроизвести потоком DataInputStream.

Метод writeDouble() записывает число double в поток, а соответствующий ему метод readDouble() затем восстанавливает его (для других типов также существуют подобные методы).

**RandomAccessFile - Чтение/запись файлов с произвольным доступом**

Работа с классом RandomAccessFile напоминает использование совмещенных в одном классе потоков DataInputStream и DataOutputStream (они реализуют те же интерфейсы DataInput и DataOutput). Кроме того, метод seek() позволяет переместиться к определенной позиции и изменить хранящееся там значение.

При использовании RandomAccessFile необходимо знать структуру файла. Класс RandomAccessFile содержит методы для чтения и записи примитивов и строк UTF-8.

RandomAccessFile может открываться в режиме чтения ("r") или чтения/записи ("rw"). Также есть режим "rws", когда файл открывается для операций чтения-записи и каждое изменение данных файла немедленно записывается на физическое устройство.

**Исключения ввода/вывода**

В большинстве случаев у классов ввода/вывода используется исключение IOException. Второе исключение FileNotFoundException передаётся в тех случаях, когад файл не может быть открыт. Данное исключение происходит от IOException, поэтому оба исключения можно обрабатывать в одном блоке catch, если у вас нет нужды обрабатывать их по отдельности.

***Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?***

OutputStreamWriter — мост между классом OutputStream и классом Writer. Символы, записанные в поток, преобразовываются в байты.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | OutputStream outputStream       = new FileOutputStream("c:\\data\\output.txt");  Writer       outputStreamWriter = new OutputStreamWriter(outputStream, "UTF-8");    outputStreamWriter.write("Hello World");    outputStreamWriter.close(); |

InputStreamReader — аналог для чтения. При помощи методов класса Reader читаются байты из потока InputStream и далее преобразуются в символы.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | InputStream inputStream       = new FileInputStream("c:\\data\\input.txt");  Reader      inputStreamReader = new InputStreamReader(inputStream, "UTF-8");    int data = inputStreamReader.read();  while(data != -1){      char theChar = (char) data;      data = inputStreamReader.read();  }    inputStreamReader.close(); |