# 17. Потоки ввода-вывода в Java. Байтовые и символьные потоки.

Объект, из которого можно считать данные, называется **потоком ввода**, а объект, в который можно записывать данные, - **потоком вывода**. Например, если надо считать содержание файла, то применяется поток ввода, а если надо записать в файл - то поток вывода.

В основе всех классов, управляющих потоками байтов, находятся два абстрактных класса: **InputStream** (представляющий потоки ввода) и **OutputStream** (представляющий потоки вывода)

Но поскольку работать с байтами не очень удобно, то для работы с потоками символов были добавлены абстрактные классы **Reader** (для чтения потоков символов) и **Writer** (для записи потоков символов).

InputStream имеет следующий абстрактный метод:

abstract int read();

Этот метод читает один байт и возвращает считанный байт или -1 если обнаружен конец источника

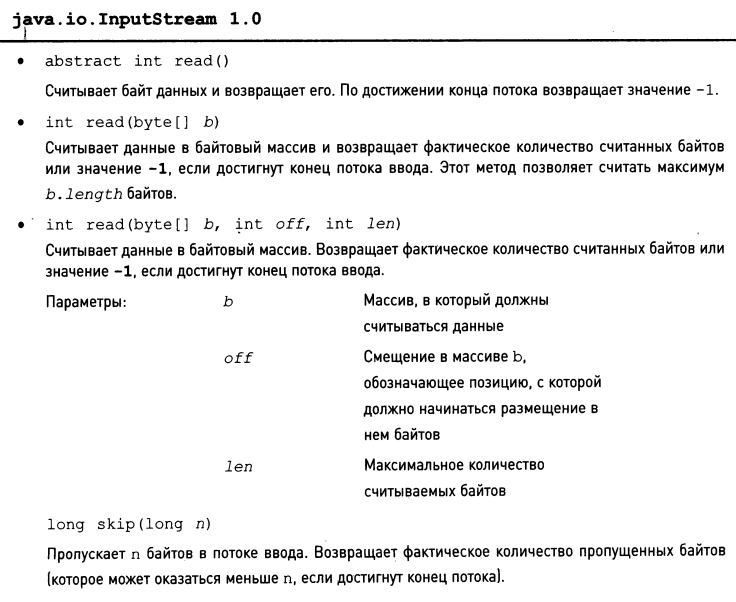
OutputStream имеет следующий абстрактный метод:

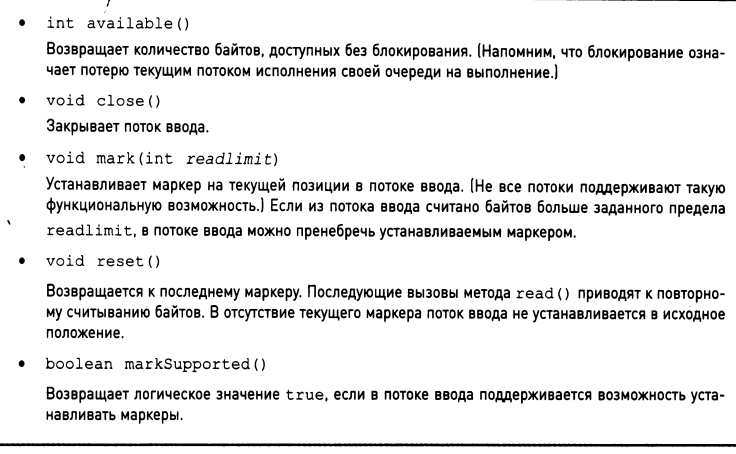
abstract void write(int b);

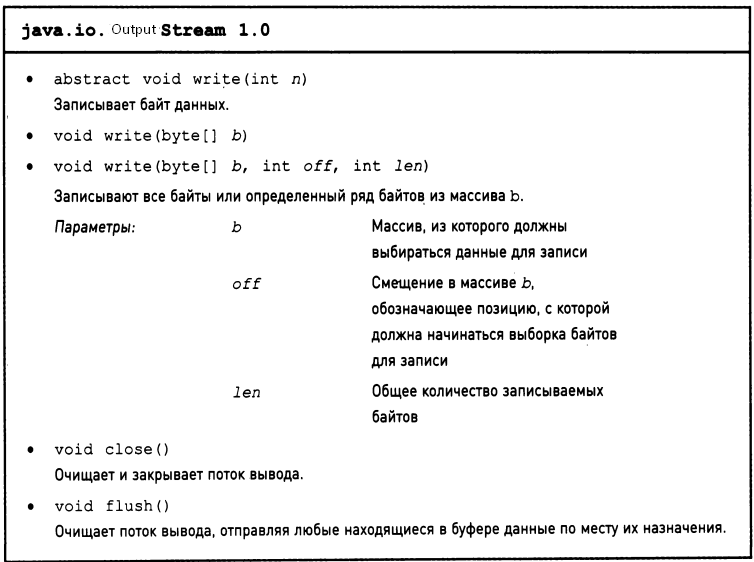
Как и метод read(), write() блокирует доступ до тех пор , пока байты не будут фактически записаны. Т.е если потоку ввода-вывода не получилось получить запрашиваем данные немедленно, то блокируется текущий поток исполнения.

Available() позволяет проверить количество байтов, доступных для считывания.

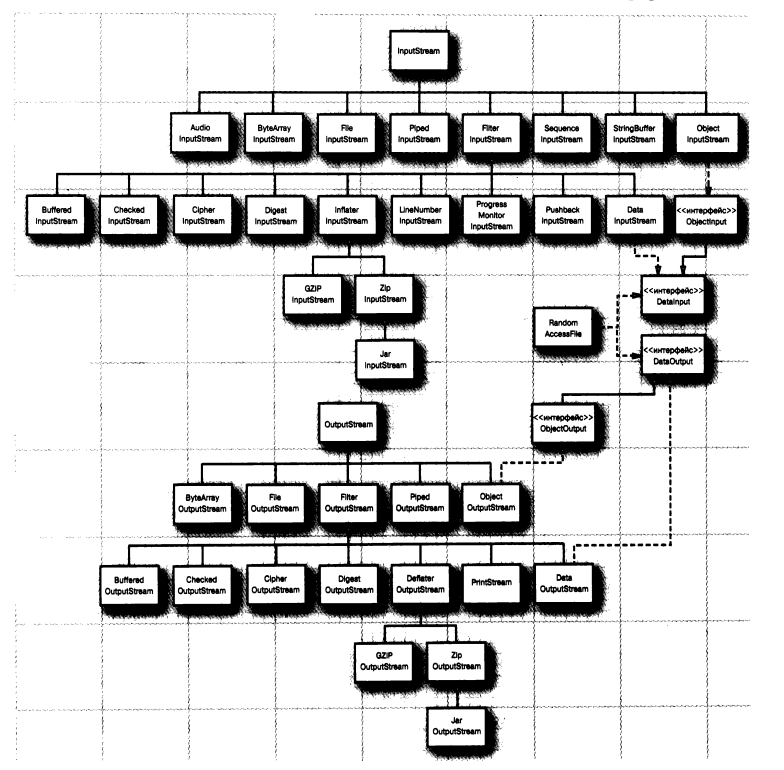
После использования потока чтения или записи следует закрыть его, используя метод close(); Такой вызов приведет к очистке системных ресурсов, чрезмерное использование которых приводит к их исчерпанию. Close() также очищает используемый для него буфер, а все символы которые размещались в нем с целью дальнейшей отправки в виде более крупного пакета данных , рассылаются по местам своего назначения. Т.е если не закрыть поток можно потерять данные. Очистить буфер от выводимых данных можно с помощью метода flush()







В Java есть комплект, состоящий из более чем 60 различных потоков ввода-вывода.



С другой стороны, для ввода-вывода текста в уникоде необходимо обращаться к подклассам таких абстрактных классов как Reader и Writer

Их базовые методы похожи на базовые методы из InputStream и OutputStream.

Метод read() возвращает кодовую единицу в уникоде( в виде целого числа от 0 до 65535) или -1, если достигнут конец. А метод write() вызывается с заданной кодовой единицей в уникоде.

Также имеются четыре дополнительный интерфейса: Closeable, Flushable,Readable и Appendable.

Closeable: void close()

Flushable: void flush()

Flushable реализуют OuputStream и Writer

Readable: int read(CharBuffer cb)

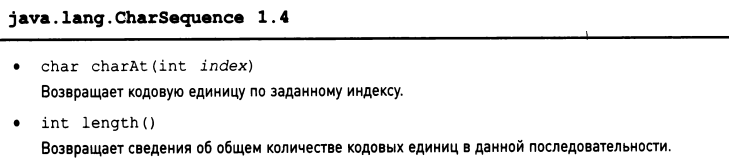
В классе CharBuffer методы для чтения и записи с последовательными и произвольным доступом. Этот класс представляет буфер в оперативной памяти или отображаемый в памяти файл.

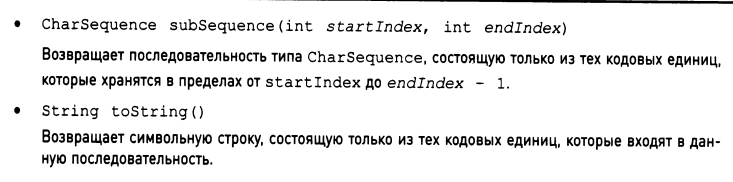
Appendable: Appendable append(char c)

Appendable append(CharSequence s)

Методы позволяют присоединять как отдельные символы так и целые последовательности символов.

Интерфейс CharSequence описывает основные свойства последовательности значений типа char. Его реализуют такие классы как String, CharBuffer, StringBuffer и StringBuilder.





Writer реализует интерфейс Appendable.

**Сочетание потоковых фильтров.**

Классы FileInputStream и FileOutputStream позволяют создавать потоки ввода-вывода и присоединить их к конкретному файлу на диске. Имя требуемого файла и полный путь к нему указывается в конструкторе.

Путь стоит вводить используя \\ потому что \-экранирующий.

Как и в абстрактных классах InputStream и OutputStream в классах с приставкой File поддерживается чтение и запись только на уровне байтов.

В Java применяется искусный механизм для разделения двух видов обязанностей. Одни потоки ввода-вывода(FileInputStream) могут извлекать байты из файлов и более экзотических мест, а другие потоки ввода-вывода (DataInputStream и PrintWriter)- cоставлять эти байты в более полезные типы данных. **НАДО ТОЛЬКО ПОДОБРАТЬ НУЖНОЕ СОЧЕТАНИЕ**.

