**Поток –** абстракция, которая используется для чтения и записи информации. Упорядоченная последовательность данных, которым соответствует определенный источник или получатель.

Все потоки ведут себя одинаково, несмотря на отличие в физических устройствах. То есть взаимодействие как с файлом, так и с консолью (и вообще чем угодно) будет одинаковым.

Потоки можно использовать для

* Чтения и записи файлов
* Поток может быть связан с сетевым сокетом, с помощью которого можно отправлять и получать данные по сети.
* Чтение и вывод в консоль.

Потоки делятся на 2 типа:

* **Поток ввода –** поток, с помощью которого можно прочитать данные.
* **Поток вывода –** поток, с помощью которого можно записать данные.

Также они разделяются на:

* **Символьные (Используют Unicode)**
* **Байтовые**

В Java потоки реализованы в пакете java.io. В нем определены четыре абстрактных класса для работы с потоками:

* **InputStream** – байтовый поток ввода
* **OutputStream** – байтовый поток вывода
* **Reader** – символьный поток ввода
* **Writer** – символьный поток вывода.

**FileInputStream и FileOutputStream**

Класс **FileOutputStream** предназначен для записи байтов в файл и унаследован от OutputStream.

В конструктор можно передать либо путь к файлу в виде строки, либо объект File. Вторым параметром *append* можно указать, дописывать ли данные в конец файла, либо перезаписать его.

Если такого файла нет, он автоматически создастся при записи.

Для записи используется метод write(byte[] data);

Класс **FileInputStream** предназначен для считывания данных из файла и является наследником InputStream.

В конструктор можно передать путь к файлу. Если файла нет, либо он не может быть открыт – генерируется FileNotFoundException.

Для чтения используется метод read() который возвращает следующий байт. При достижении конца файла возвращается -1. Можно читать также в массив байт.

Оба класса предназначены для чтения и записи двоичных файлов.

**ByteArrayInputStream и ByteArrayOutputStream**

Используются для работы с массивами байтов.

**ByteArrayInputStream** используется для чтение данных из массива байтов. В конструктор принимает массив байтов (опционально смещение и кол-во символов).

**ByteArrayOutputStream** используется для вывода данных в массив байтов. Может принимать в конструктор размер выходного массива (по умолчанию 32 байта).

Оба потока не требуют закрытия явным образом.

**BufferedInputStream и BufferedOutputStream**

Данные классы используются для оптимизации операций ввода вывода. Они имеют специальный буфер в памяти. Они оборачивают собой обычные потоки.

**BufferedInputStream** накапливает вводимые данные в специальном буфере без постоянного обращения к устройству ввода. Имеет 2 конструктора:



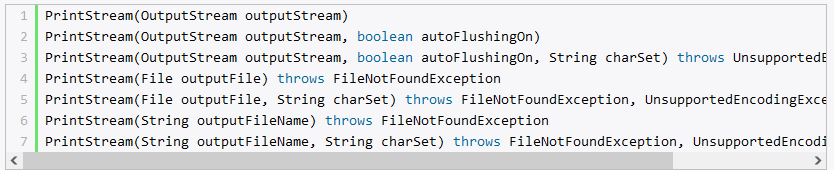
**BufferedOutputStream** создает буфер для потоков вывода. Накапливает вводимые данные, и когда буфер заполнен, производится запись данных. Конструктор аналогичен BufferedInputStream.

Аналогично работают классы **BufferedReader** и **BufferedWriter**. BufferedReader имеет метод readLine, который считывает строку целиком и возвращает String().

**PrintStream и PrintWriter**

Класс **PrintStream** используется для вывода на консоль. System.out.println() как раз использует его (out – объект класса PrintStream, а println() его метод).

Этот класс можно использовать и для записи в поток вывода, передав его в конструктор. Можно также передать объект File либо путь к файлу.



Основные методы PrintStream:

* println(): вывод строковой информации с переводом строки
* print(): вывод строковой информации без перевода строки
* printf(): форматированный вывод

**PrintWriter** можно использовать как для вывода информации на консоль, так и в файл или любой другой поток вывода. Используется для записи символьной информации. Упрощает интернационализацию программ (за счет Unicode).

**Закрытие потоков**

После завершения потоки нужно закрывать методом **close()** который определен в интерфейсе Closeable. В случае, если поток окажется не закрыт, может происходить утечка памяти.

Лучше всего закрывать потоки с помощью try-with-resources.

**Класс File**

Оперирует непосредственно файлами и каталогами. В нем не определяется каким образом читаются и пишутся данные, но описываются свойства самих файлов (Права доступа, размер, время, дата и путь к каталогу, перемещение по иерархиям подкаталогов а также манипулирования этой информацией).

С помощью данного класса можно переименовывать и удалять файлы и каталоги. В каталоге можно также создавать подкаталоги.

**Flushable**

Данный интерфейс имеет единственный метод flush(). Этот метод вызывается чтобы принудительно отправить буферизированные данные в поток вывода (записать их на физическое устройство).