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()

java.io.InputStream 1.0

- `abstract int read()`

  Считывает байт данных и возвращает его. По достижении конца потока возвращает значение `-1`.

- `int read(byte[] b)`

  Считывает данные в байтовый массив и возвращает фактическое количество считанных байтов или значение **-1**, если достигнут конец потока ввода. Этот метод позволяет считать максимум *b.length* байтов.

- `int read(byte[] b, int off, int len)`

  Считывает данные в байтовый массив. Возвращает фактическое количество считанных байтов или значение **-1**, если достигнут конец потока ввода.

Параметры:		
	b	Массив, в который должны считываться данные
	off	Смещение в массиве `b`, обозначающее позицию, с которой должно начинаться размещение в нем байтов
	len	Максимальное количество считываемых байтов

  `long skip(long n)`

  Пропускает `n` байтов в потоке ввода. Возвращает фактическое количество пропущенных байтов (которое может оказаться меньше `n`, если достигнут конец потока).

- `int available()`

  Возвращает количество байтов, доступных без блокирования. (Напомним, что блокирование означает потерю текущим потоком исполнения своей очереди на выполнение.)

- `void close()`

  Закрывает поток ввода.

- `void mark(int readlimit)`

  Устанавливает маркер на текущей позиции в потоке ввода. (Не все потоки поддерживают такую функциональную возможность.) Если из потока ввода считано байтов больше заданного предела `readlimit`, в потоке ввода можно пренебречь устанавливаемым маркером.

- `void reset()`

  Возвращается к последнему маркеру. Последующие вызовы метода `read()` приводят к повторному считыванию байтов. В отсутствие текущего маркера поток ввода не устанавливается в исходное положение.

- `boolean markSupported()`

  Возвращает логическое значение `true`, если в потоке ввода поддерживается возможность устанавливать маркеры.

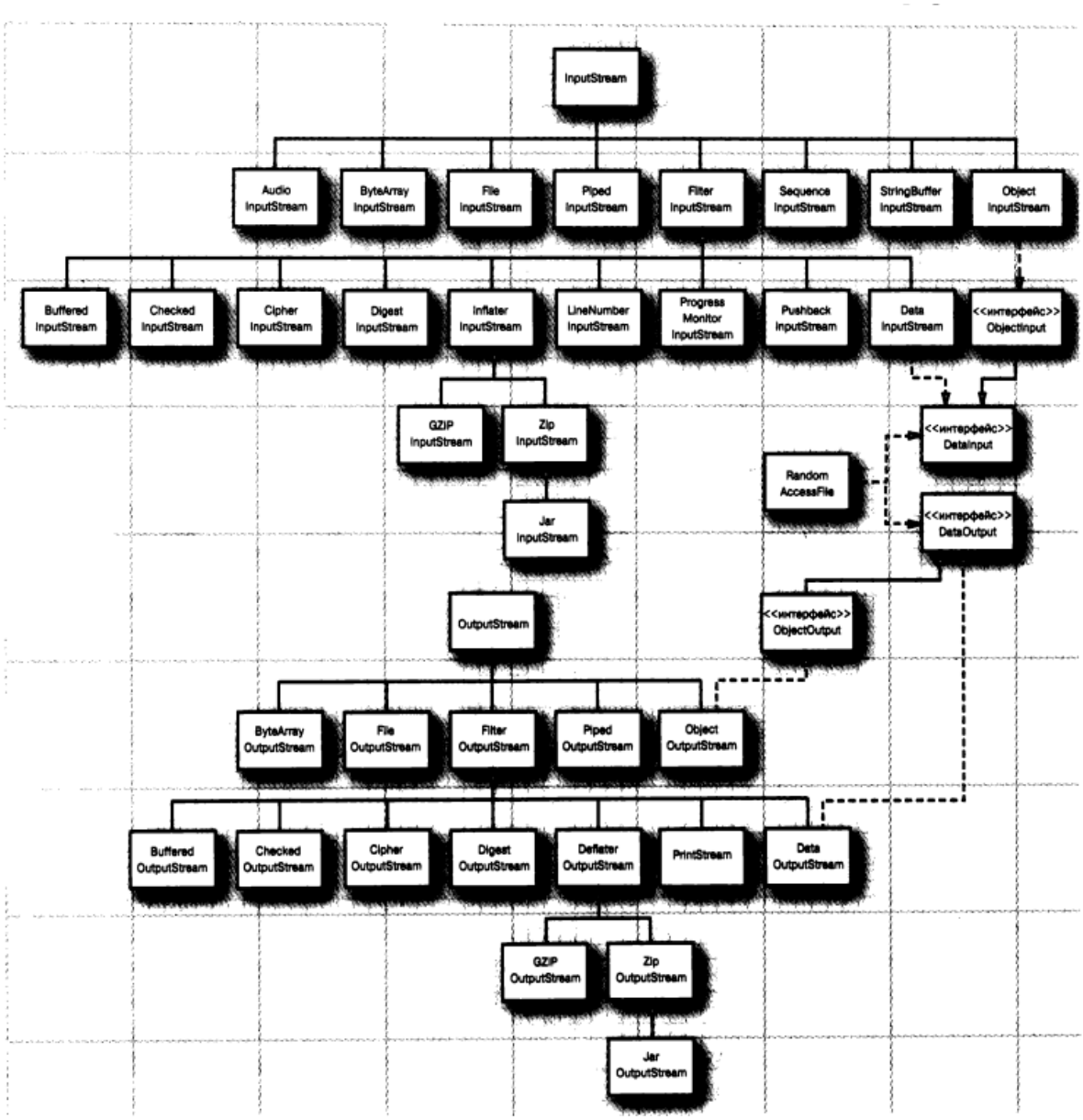
java.io. Output **Stream 1.0**

- `abstract void write(int n)`
  Записывает байт данных.
- `void write(byte[] b)`
- `void write(byte[] b, int off, int len)`
  Записывают все байты или определенный ряд байтов из массива `b`.

Параметры:		
	b	Массив, из которого должны выбираться данные для записи
	off	Смещение в массиве *b*, обозначающее позицию, с которой должна начинаться выборка байтов для записи
	len	Общее количество записываемых байтов

- `void close()`
  Очищает и закрывает поток вывода.
- `void flush()`
  Очищает поток вывода, отправляя любые находящиеся в буфере данные по месту их назначения.

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.

java.lang.CharSequence 1.4

- `char charAt(int index)`
  Возвращает кодовую единицу по заданному индексу.
- `int length()`
  Возвращает сведения об общем количестве кодовых единиц в данной последовательности.
- `CharSequence subSequence(int startIndex, int endIndex)`
  Возвращает последовательность типа CharSequence, состоящую только из тех кодовых единиц, которые хранятся в пределах от startIndex до *endIndex* - 1.
- `String toString()`
  Возвращает символьную строку, состоящую только из тех кодовых единиц, которые входят в данную последовательность.

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)- составлять эти байты в более полезные типы данных. **НАДО ТОЛЬКО ПОДОБРАТЬ НУЖНОЕ СОЧЕТАНИЕ.**

java.io.FileInputStream 1.0

- `FileInputStream(String name)`
- `FileInputStream(File file)`

Создают новый поток ввода из файла, путь к которому указывается в символьной строке `name` или в объекте *file*. (О классе `File` более подробно будет рассказываться в конце этой *главы*.) Если указываемый путь не является абсолютным, он определяется относительно рабочего каталога, который был установлен при запуске виртуальной машины.

- `FileOutputStream(String name)`
- `FileOutputStream(String name, boolean append)`
- `FileOutputStream(File file)`
- `FileOutputStream(File file, boolean append)`

Создают новый поток вывода в файл, который указывается в символьной строке `name` или в объекте *file*. (О классе `File` более подробно будет рассказываться в конце этой главы.) Если параметр `append` принимает логическое значение `true`, данные выводятся в конец файла, а если обнаружится уже существующий файл с таким же именем, он не удаляется. В противном случае удаляется любой уже существующий файл с таким же именем.

java.io.BufferedInputStream 1.0

- `BufferedInputStream(InputStream in)`

Создает буферизированный поток ввода. Такой поток способен накапливать вводимые байты без постоянного обращения к устройству ввода. Когда буфер пуст, в него считывается новый блок данных из потока.

java.io.BufferedOutputStream 1.0

- `BufferedOutputStream(OutputStream out)`

Создает буферизированный поток вывода. Такой поток способен накапливать выводимые байты без постоянного обращения к устройству вывода. Когда буфер заполнен или поток очищен, данные записываются.

java.io.PushbackInputStream 1.0

- `PushbackInputStream(InputStream in)`
- `PushbackInputStream(InputStream in, int size)`

  Создают поток ввода или вывода с однобайтовым буфером для чтения с упреждением или буфером указанного размера для возврата данных обратно в поток.

- `void unread(int b)`

  Возвращает байт обратно в поток, чтобы он мог быть извлечен снова при последующем вызове для чтения.

  Параметры: `b` Повторно читаемый байт