**InputStream/OutputStream**

[Java Core](https://javarush.ru/quests/QUEST_JAVA_CORE)

[Уровень 8](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=8), Лекция 4

— Привет, Амиго! Сегодня мы снова будем заниматься разбором работы **InputStream** и **OutputStream**. На самом деле, то первое объяснение было немного упрощенным. Это не интерфейсы, а абстрактные классы, и они даже имеют по паре реализованных методов. Давай посмотрим, какие методы у них есть:

|  |  |
| --- | --- |
| **Методы класса InputStream** | **Что метод делает** |
| int read(byte[] buff); | — метод сразу читает блок байт в буфер (массив байт), пока буфер не заполнится или не закончатся байты там, откуда он их читает. Метод возвращает количество реально прочитанных байт (оно может быть меньше длины массива) |
| int read(); | — метод читает один байт и возвращает его как результат. Результат расширяется до int, для красоты. Если все байты уже прочитаны, метод вернет «-1». |
| int available(); | — метод возвращает количество непрочитанных (доступных) байт. |
| void close(); | — метод «закрывает» поток – вызывается после окончания работы с потоком. Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д. Из потока больше нельзя читать данные. |

— Т.е. мы можем читать не только по одному байту, а и целыми блоками?

— Да.

— А записывать целыми блоками тоже можно?

— Да, вот смотри:

|  |  |
| --- | --- |
| **Методы OutputStream** | **Что метод делает** |
| void write(int c); | — метод записывает один байт информации. Тип int сужается до byte, лишняя часть просто отбрасывается. |
| void write(byte[] buff); | — метод записывает блок байт. |
| void write(byte[] buff, int from, int count); | — метод записывает часть блока байт. Используется в случаях, когда есть вероятность, что блок данных был заполнен не целиком |
| void flush(); | — если есть данные, которые хранятся где-то внутри и еще не записаны, то они записываются. |
| void close(); | — метод «закрывает» поток – вызывается после окончания работы с потоком. Объект выполняет служебные операции, связанные с закрытием файла на диске и т.д.В поток больше нельзя писать данные, flush при этом вызывается автоматически. |

— А как будет выглядеть код копирования файла, если мы будем читать не по одному байту, а целыми блоками?

— Гм. Примерно так:

Копируем файл на диске

public static void main(String[] args) throws Exception

{

//Создаем поток-чтения-байт-из-файла

FileInputStream inputStream = new FileInputStream("c:/data.txt");

// Создаем поток-записи-байт-в-файл

FileOutputStream outputStream = new FileOutputStream("c:/result.txt");

byte[] buffer = new byte[1000];

while (inputStream.available() > 0) //пока есть еще непрочитанные байты

{

// прочитать очередной блок байт в переменную buffer и реальное количество в count

int count = inputStream.read(buffer);

outputStream.write(buffer, 0, count); //записать блок(часть блока) во второй поток

}

inputStream.close(); //закрываем оба потока. Они больше не нужны.

outputStream.close();

}

— С буфером все понятно, а что это за переменная count?

— Когда мы читаем самый последний блок данных в файле, может оказаться, что байт осталось не 1000, а, скажем, 328. Тогда и при записи нужно указать, что записать не весь блок, а только его первые 328 байт.

Метод read при чтении последнего блока вернет значение равное количеству реально прочитанных байт. Для всех чтений – 1000, а для последнего блока – 328.

Поэтому при записи блока мы указываем, что нужно записать не все байты из буфера, а только 328 (т.е. значение, хранимое в переменной count).

— Теперь понятно, как все это работает. Спасибо, Элли.