**Потоки для ввода/вывода файлов**

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

[Уровень 8](https://javarush.ru/quests/lectures?quest=QUEST_JAVA_CORE&level=8), Лекция 2

— А начнем мы с потоков для ввода/вывода файлов. Но обо всем по порядку.

Для чтений и записи файлов есть два класса:  **FileInputStream** и **FileOutputStream**. Как ты уже, наверное, догадался,  **FileInputStream** позволяет последовательно читать из файла байты, а **FileOutputStream** – записывать в файл байты. Вот какие методы есть у этих классов:

|  |  |
| --- | --- |
| **Метод** | **Что метод делает** |
| FileInputStream(String fileName); | — это конструктор. Позволяет указать имя файла на диске, из которого созданный объект будет читать данные. |
| int read(); | — метод читает один байт из файла и возвращает его как результат. Тип результата расширяется до int. |
| 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. |

Давай ради интереса посчитаем сумму всех байт в файле на диске. Вот как будет выглядеть этот код:

Подсчет суммы всех байт файла на диске

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

{

//создаем объект FileInputStream, привязанный к файлу «c:/data.txt».

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

long sum = 0;

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

{

int data = inputStream.read(); //прочитать очередной байт

sum += data; //добавить его к общей сумме

}

inputStream.close(); // закрываем поток

System.out.println(sum); //выводим сумму на экран.

}

— Мы уже раньше что-то подобное разбирали. А как устроен FileOutputStream?

— Ок. Вот, смотри:

|  |  |
| --- | --- |
| **Метод** | **Что метод делает** |
| FileOutputStream (String fileName); | — это конструктор. Позволяет указать имя файла на диске, в который созданный объект будет писать данные. |
| void write(int data); | — метод записывает очередной байт, обрезая переменную data до одного байта. |
| void flush(); | — часто данные для записи сначала собираются в большие блоки в памяти, а потом только пишутся на диск.  Команда flush требует немедленно записать всю несохраненную информацию на диск. |
| 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. |

— И все?

— Да, тут фактически только один метод для записи – write, который записывает только один байт за раз. Но благодаря ему можно записать в файл сколько угодно информации.

Программирование – это процесс разбиения одной большой и сложной задачи на много маленьких. Тут происходит практически тот же процесс: чтение и запись больших данных маленькими порциями – по кусочкам – по одному байту.

Вот как можно скопировать файл на диске, пользуясь этими классами:

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");

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

{

int data = inputStream.read(); // прочитать очередной байт в переменную data

outputStream.write(data); // и записать его во второй поток

}

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

outputStream.close();

}

— Спасибо, Риша. Наконец-то понял, как на самом деле работает этот код.