Пост

Лента постов Все группы Мои группы

Professor Hans Noodles

41 уровень

19 марта 2018 16:00 6843 8

Считывание с клавиатуры — «ридеры»

Пост из группы Java Developer

3785 участников

Присоединиться

Привет!

В лекциях и задачах 3 уровня мы научились выводить данные в консоль, и наоборот — считывать данные с клавиатуры.

Ты даже научился использовать для этого сложную конструкцию:

```
1  BufferedReader reader = new BufferedReader(new InputStreamReader(Syste
```

Но на один вопрос мы так и не ответили.

А как это вообще работает?

На самом деле, любая программа чаще существует не сама по себе. Она может общаться с другими программами, системами, интернетом и т.д.

Под словом “общаться” мы в первую очередь подразуемеваем “обмениваться данными”. То есть, принимать какие-то данные извне, а собственные данные — наоборот, куда-то отправлять.

Примеров обмена данными между программами много даже в повседневной жизни.

Так, на многих сайтах ты можешь вместо регистрации авторизоваться при помощи своего аккаунта в Facebook или Twitter. В этой ситуации две программы, скажем, Twitter и сайт, на котором ты пытаешься зарегистрироваться, обмениваются необходимыми данными между собой, после чего ты видишь конечный результат — успешную авторизацию.

Для описания процесса обмена данными в программировании часто используется термин “**поток**”.

Откуда вообще взялось такое название? “Поток” больше ассоциируется с рекой или ручьем, чем с программированием. На самом деле, это неспроста:)

Поток — это, по сути, перемещающийся кусок данных. То есть в программировании по потоку “течет” не вода, а данные в виде байтов и символов.

Из потока данных мы можем получать данные частями и что-то с ними делать.

Опять же, применим "водно-текучую" аналогию: из реки можно зачерпнуть воды, чтобы сварить суп, потушить пожар или полить цветы.

При помощи потоков ты можешь работать с любыми источниками данных: интернет, файловая система твоего компьютера или что-то еще — без разницы. Потоки — инструмент универсальный. Они позволяют программе получать данные отовсюду (входящие потоки) и отправлять их куда угодно (исходящие). Их задача одна — брать данные в одном месте и отправлять в другое.

Потоки делятся на два вида:

1. Входящий поток (**Input**) — используется для приема данных

2. Исходящий поток (**Output**) — для отправки данных.

Входящий поток данных в Java реализован в классе `InputStream`, исходящий — в классе `OutputStream`.

Но есть и другой способ деления потоков. Они делятся не только на входящие и исходящие, но также на **байтовые** и **символьные**. Тут смысл понятен и без разъяснений: байтовый поток передает информацию в виде набора байт, а символьный — в виде набора символов.

В этой лекции мы подробно остановимся на входящих потоках. А информацию про исходящие я приложу ссылки в конце, и ты сможешь прочитать об этом сам:)

Итак, наш код:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(Syste
```

Ты, наверное, еще во время чтения лекций подумал, что выглядит это довольно устрашающе?:) Но это только до тех пор, пока мы не разобрались, как эта штука работает. Сейчас исправим!

Начнем с конца.

`System.in` — это объект класса `InputStream`, о котором мы говорили в начале. Это входящий поток, и он привязан к системному устройству ввода данных — клавиатуре.

Вы с ним, кстати, косвенно знакомы. Ведь ты часто используешь в работе его “коллегу” — `System.out`! `System.out` — это системный поток **вывода** данных, он используется для вывода на консоль в том самом методе `System.out.println()`, которым ты постоянно пользуешься:)

`System.out` — поток для отправки данных на консоль, а `System.in` — для получения данных с клавиатуры. Все просто:)

Более того: чтобы считать данные с клавиатуры, мы можем обойтись без этой большой конструкции и написать просто: `System.in.read()`;

```
public class Main {

    public static void main(String[] args) throws IOException {

        while (true) {
            int x = System.in.read();
```

```
            System.out.println(x);
        }
    }
}
```

В классе `InputStream` (а `System.in`, напомню, является объектом класса `InputStream`) есть метод `read()`, который позволяет считывать данные.

Одна проблема: он считывает **байты**, а не **символы**. Попробуем считать с клавиатуры русскую букву “Я”.

Вывод в консоль:

Я
208
175
10

Русские буквы занимают в памяти компьютера 2 байта (в отличие от английских, которые занимают всего 1). В данном случае из потока считалось 3 байта: два первых обозначают нашу букву “Я”, и еще один — перенос строки (Enter).

Поэтому вариант использовать “голый” `System.in` нам не подойдет.

Человек (за редкими исключениями!) не умеет читать байты. Тут-то нам на помощь и приходит следующий класс — **`InputStreamReader`**!

Давай разберемся, что это за зверь такой.

```
BufferedReader bufferedReader = new BufferedReader(new InputStreamRead
```

Мы передаем поток `System.in` объекту **`InputStreamReader`**.

В общем-то, если перевести его название на русский, все выглядит очевидно — “считыватель входящих потоков”.

Собственно, именно для этого он и нужен! Мы создаем объект класса `InputStreamReader` и передаем ему входящий поток, из которого он должен считывать данные. В данном случае...

```
new InputStreamReader(System.in)
```

...мы говорим ему — “ты будешь считывать данные из системного входящего потока (с клавиатуры)”.

Но это не единственная его функция!

`InputStreamReader` не только получает данные из потока. Он еще и **преобразует байтовые потоки в символьные**. Иными словами, тебе уже не нужно самому заботиться о переводе считанных данных с “компьютерного” языка на “человеческий” — `InputStreamreader` сделает все за тебя.

`InputStreamReader`, конечно, может читать данные не только из консоли, но и из других мест. Например, из файла:

```
1  import java.io.FileInputStream;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4
5  public class Main {
6
7      public static void main(String[] args) throws IOException {
8          InputStreamReader inputStreamReader = new InputStreamReader(new
9      }
10 }
```

Здесь мы создали входящий поток данных `FileInputStream` (это одна из разновидностей `InputStream`), передали в него путь к файлу, а сам поток передали `InputStreamReader`’у.

Теперь он сможет читать данные из этого файла (если файл по этому пути существует, конечно).

Для чтения данных (неважно откуда, из консоли, файла или откуда-то еще) в классе `InputStreamReader` тоже используется метод `read()`.

В чем же разница между **`System.in.read()`** и **`InputStreamReader.read()`**?

Давай попробуем считать ту же самую букву “Я” с помощью **`InputStreamReader`**.

Напомню, вот что считал `System.in.read()`:

JavaRush
Я
208

175

10

А как ту же самую работу проделает `InputStreamReader`?

```
1  public class Main {
2
3     public static void main(String[] args) throws IOException {
4
5         InputStreamReader reader = new InputStreamReader(System.in);
6         while (true) {
7             int x = reader.read();
8             System.out.println(x);
9         }
10    }
11 }
```

Вывод в консоль:

Я

1071

10

Разница видна сразу. Последний байт — для переноса строки — остался без изменений (число 10), а вот считанная буква “Я” была преобразована в единый код “1071”. Это и есть считывание по символам!

Если вдруг не веришь, что код 1071 обозначает букву “Я” — в этом легко убедиться:)

```
1  import java.io.IOException;
2
3  public class Main {
4
5     public static void main(String[] args) throws IOException {
6
7         char x = 1071;
8         System.out.println(x);
9     }
10 }
```

Вывод в консоль:

Я

Но если `InputStreamReader` так хорош — зачем нужен еще **`BufferedReader`**? `InputStreamReader` умеет и считывать данные, и конвертировать байты в символы — а что нам еще нужно-то? Зачем еще один Reader? :/

Ответ очень прост — для большей **производительности** и большего **удобства**.

Начнем с производительности.

`BufferedReader` при считывании данных использует специальную область — буфер, куда “складывает” прочитанные символы. В итоге, когда эти символы понадобятся нам в программе — они будут взяты из буфера, а не напрямую из источника данных (клавиатуры, файла и т.п.), а это экономит очень много ресурсов.

Чтобы понять как это работает — представь, для примера, работу курьера в крупной компании.

Курьер сидит в офисе и ждет, когда ему принесут посылки на доставку.

Каждый раз, получив новую посылку, он может сразу же отправляться в дорогу. Но посылок в течение дня может быть много, и ему придется каждый раз мотаться между офисом и адресами.

Вместо этого курьер поставил в офисе коробку, куда все желающие складывают свои посылки. Теперь курьер может спокойно взять коробку и отправляться по адресам — он сэкономит очень много времени, ведь ему не придется каждый раз возвращаться в офис.

Коробка в этом примере как раз является буфером, а офис — источником данных. Курьеру намного проще при доставке брать письмо из общей коробки, чем каждый раз ехать в офис. Еще и бензин сэкономит. Так же и в программе — гораздо менее затратно по ресурсам брать данные из буфера, а не обращаться всякий раз к источнику данных.

Поэтому **`BufferedReader`** + **`InputStreamReader`** **работает быстрее, чем просто** **`InputStreamReader`**.

С производительность разобрались, а что с удобством?

Главный плюс в том, что `Bufferedreader` умеет читать данные не только по одному символу (хотя метод `read()` для этих целей у него тоже есть), а еще и целыми строками! Делается это с помощью метода `readLine()`;

```
1  public class Main {
2
```

```
3     public static void main(String[] args) throws IOException {
4
5         BufferedReader reader = new BufferedReader(new InputStreamReade
6         String s = reader.readLine();
7         System.out.println("Мы считали с клавиатуры эту строку:");
8         System.out.println(s);
9     }
10  }
```

Вывод в консоль:

JavaRush — лучший сайт для изучения Java!
Мы считали с клавиатуры эту строку:
JavaRush — лучший сайт для изучения Java!

Это особенно удобно в случае чтения большого объема данных. Одну-две строчки текста еще можно считать посимвольно. А вот считать “Войну и мир” по одной букве будет уже несколько проблематично:)

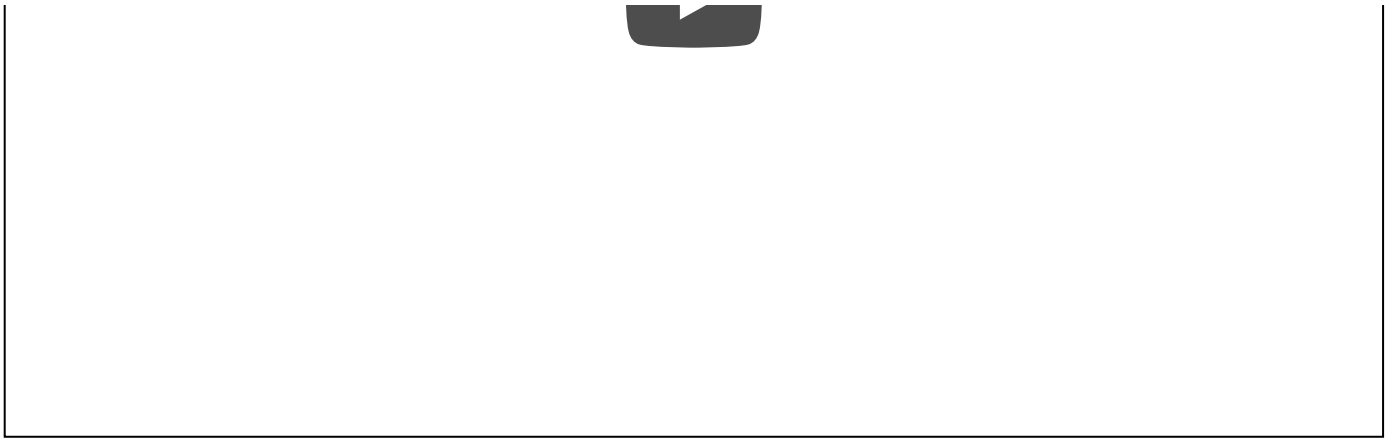
Теперь работа потоков стала гораздо более понятной для тебя. Для дальнейшего изучения — вот тебе несколько ссылок для самостоятельного чтения:

http://java-online.ru/java-inputstream.xhtml
http://java-online.ru/java-outputstream.xhtml

Здесь ты можешь прочитать подробнее про входящие и исходящие потоки. Обрати особое внимание на таблицы с методами этих классов. Мы будем рассматривать их в будущем, но ознакомиться можно уже сейчас.

Видеообзор `BufferedReader`’а от одного из наших учеников. Да-да, наши ученики не только учатся сами, но и записывают обучающие видео для других! Не забудь поставить лайк и подписаться на наш канал:)

Загадочный BufferedReader [видео от учеников JavaRush]



https://javarush.ru/quests/lectures/questcore.level09.lecture06 — одна из лекций JavaRush, посвященная `BufferedReader` и `InputStreamReader`

https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html
https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html — документация Oracle о классах `BufferedReader` и `InputStreamReader`.

Лучше с самого начала учебы приучать себя к чтению официальной документации. Она является главным источником знаний по языку, и большинство ответов всегда можно найти там.

У нас же нет в этом случае буфера введенных символов.

Обьяснение логично для файлов, но с клавиатурой у меня диссонанс :)

Ответить 0

Макс 19 уровень, Киев
22 октября, 01:37

```java
import java.io.IOException;
import java.io.InputStreamReader;

public class Solution {
    public static void main(String[] args) throws IOException {
        InputStreamReader reader = new InputStreamReader(System.in)

        System.out.println(reader.getEncoding()); // Возвращает имя
        System.out.println(reader.ready()); // Сообщает, остались л

        // Считывает с клавиатуры байтовые данные и преобразует их
        do {
            int x1 = reader.read(); // Вводим сколько угодно символ
            System.out.println(reader.ready()); // Сообщает, остали
            System.out.println(x1); // Выводит символьные данные (н
        } while ((reader.ready())); // Цикл работает, пока не будут

        // Массив для хранения считываемых данных. В каждую ячейку
        char[] chars = new char[3];

        // Считываемые символы сохраняет в массив (1 ячейка = 1 сим
        // Символы читает по-порядку, начиная с первого
        // Сохраняет в массив начиная с ячейки под индексом, которы
        // "length" - сколько максимум символов будет прочитано и с
        // ВАЖНО!!! Параметр "length" должен быть <= размера массив
        int x2 = reader.read(chars, 0, 3); // x2 - хранит КОЛИЧЕСТВ

        System.out.println(reader.ready()); // Сообщает, остались л
        System.out.println("К-во символов, занесённых в массив: " +

        System.out.println("\nСодержимое массива:");
        // Выводит содержимое массива
        for (char i: chars) {
            System.out.println(i);
        }
    }
}
```

Ответить +2

Dronya_33 9 уровень, Москва
воскресенье, 14:42

Дружище, у тебя заполнение массива пропало. Для меня не беда, а новичкам вряд ли будет понятно. А так молодец!

Ответить 0

Eugene Bass 10 уровень
30 сентября, 14:17

Почему при использовании System.in.read(); в Eclipse результатом вывода на экран являются совершенно другие данные
Я
223
13
10
?
Хотя при перекодировке через InputStreamReader(System.in); результат в Eclipse становится таким же за исключением дополнительного кода 13 (который вроде бы как уже не используется) - выводит
Я
1071
13
10

Ответить 0

Nevendaar 12 уровень, Минск
3 октября, 17:22

Я могу сделать предположение. Видите ли, в видео автор говорит про кодировку. Я в типах и способах кодировки не силён. Но их точно несколько. В эклипсе возможно используется другая кодировка. Вот и разные результаты. А код 1071 это, как говорилось, код символа. Он должен быть одинаков везде, чтобы при выводе char Symbol = 1071 выводилось на экран "Я".
Но это лишь мои дилетантские мысли.

Ответить 0

Андрей 8 уровень
22 октября, 01:41

насколько я понимаю это может зависеть еще от операционной системы и кодировки по умолчанию

Ответить 0

Eugene Bass 10 уровень
22 октября, 17:09

На одной и той-же ОС

Ответить 0

Ринат 5 уровень, Екатеринбург
24 сентября, 22:22

Ссылка на шестую лекцию из девятого уровня недоступна (пришел по ссылке из третьего уровня)

Ответить +1

Программистами не рождаются