Theory: Input streams

1 hour 0 / 0 problems solved

Skip this topic Start practicing

4000 users solved this topic. Latest completion was **about 19 hours ago**.

As a rule, every program consumes some data as a way to communicate with the outer world. It can be a user input from the console, a configuration file, or something else. Java has a common mechanism for consuming data called **input streams**. We have already provided some text as data for our programs, and now it is time to dive deeper and explore the whole mechanism.

3 required topics

✓ What are streams In project

✓ Writing files In project

✓ Array 13 In project

3 dependent topics

Serialization basics In project

Try with resources

Sockets

§1. Sources

Data can be obtained from many points considered as providers. Besides standard input or files, those can be network connections, in-memory buffers, or even objects. All of them are called **sources** for input streams. In fact, a source is any data that can be consumed and processed by a program. Working with data is quite a specific thing, and each source needs a specialized class.

§2. Character streams

There are several classes for reading text. They are called character input streams and allow reading text data: `char` or `String`. For instance, there are `FileReader`, `CharArrayReader`, `StringReader`, etc.

The class name indicates what type of source it uses as input and usually ends with *Reader,* since *all* such classes extend the `java.io.Reader` class.

Each class provides a set of useful methods while they also have common methods for reading data:

- `int read()` reads a single character. If the end of the stream is reached, the method returns the value `-1`. Otherwise, it returns the numerical representation of the character according to the current encoding;
- `int read(char[] cbuf)` reads a sequence of characters into the passed array up to its capacity and returns the number of characters that were actually read. It can also return `-1` in case no data was read;
- `int read(char[] cbuf, int off, int len)` reads characters into a portion of an array.

These methods return the number of characters that were actually read or `-1`. They also block the program from running until some input is available or the end of the stream is reached.

Another important method is `void close()` that should be invoked after a stream was used.

If you're familiar with the try-with-resources construction, you know it is a better way to prevent resource leaks. For now, we're skipping it for learning purposes.

§3. Example of a character stream

Let's consider `FileReader` as an example of the `Reader` classes. `FileReader` has a set of constructors. Here are some of them:

- `new FileReader(String fileName)`
- `new FileReader(String fileName, Charset charset)`
- `new FileReader(File file)`
- `new FileReader(File file, Charset charset)`

As you can see, it can read text data from the file indicated either by a path `String` or as a `File` object.

A charset is a class that declares the encoding from sequences of bytes to characters. By default, java uses the UTF-16 encoding, suitable for most tasks. However, sometimes the file may have another encoding and you'll have to use a different charset to read the content of the file properly.

Now let's try to read a file. Say we have a file `file.txt` with the following content: `input stream`.

```
Reader reader = new FileReader("file.txt");

char first = (char) reader.read(); // i
char second = (char) reader.read(); // n

char[] others = new char[12];
int number = reader.read(others); // 10
```

After running the code, `others` will contain `['p', 'u', 't', ' ', 's', 't', 'r', 'e', 'a', 'm', '\u0000', '\u0000']`.

Let's explain the result. Since we've read the first two letters into other variables, the first 10 characters of `others` are filled starting from the third letter. When the stream reached the end of the file it stopped reading, so the last two characters are not updated.

When you create an empty array it is actually filled with default values, which are `'\u0000'` for a char array. This is the reason why the last 2 elements of `others` are `'\u0000'`.

The tricky thing here is that `'\u0000'` is interpreted as an empty symbol and not displayed at all, although technically it is present. Remember that when you read data into an array.

Another common way of reading a text data stream is to read it char by char until the stream is closed:

```
FileReader reader = new FileReader("file.txt");

int charAsNumber = reader.read();
while(charAsNumber != -1) {
    char character = (char) charAsNumber;
    System.out.print(character);
    charAsNumber = reader.read();
}
reader.close();
```

When `-1` is returned, it means the end of stream was reached, so that is there's nothing left to read.

Next section

5 sections left Expand all

Table of contents: