# Android bufferedreader readline returns null

The Java BufferedReader class, java.io.BufferedReader, provides buffering for Java Reader instances. Buffering can speed up IO quite a bit. Instead of reading one character from the main reader, Java BufferedReader reads a larger block (array) at a time. This is usually much faster, especially for access to the drive and large amounts of data. Java BufferedReader is similar to BufferedInputStream, but they are not quite the same. The main difference between BufferedReader and BufferedInputStream is that BufferedReader reads characters (text), while BufferedInputStream reads raw bytes. The Java BufferedReader class is a Java Reader class, so you can use BufferedReader wherever you want a Reader. Example Java BufferedReader To add buffering to a Java Reader instance, simply wrap it up in BufferedReader. Here's what it looks like: BufferedReader BufferedReader is the new BufferedReader (c:datainput-file.txt); This example creates a BufferedReader that wraps FileReader. BufferedReader will read a block of characters from FileReader (usually in an array of characters). Thus, each character returned from the reading () returns from this inner array. When the array reads fully bufferedReader reads a new data block in the array, etc. BufferedReader Buffer Size You can set the buffer size for internal use of BufferedReader. You provide size as a constructor as it is: Int bufferSize Nos. 8 and 1024; BufferedReader BufferedReader - new BufferedReader (new FileReader (c:datainput-file.txt), bufferSize); This example sets an internal buffer of up to 8 KB. It is best to use buffer sizes, multiples of 1024 bytes. This works best with most built-in buffering in hard drives, etc. Except for adding buffering to Reader instances, Java BufferedReader behaves almost like a reader. BufferedReader has one additional method though, readLine () method. This method can be useful if you need to read one line at a time. Here's an example of BufferedReader readLine: Line - bufferedReader.readLine(); The readLine method returns the text line (the entire text until a line break is found) read from BufferedReader. If there is no more reading data from the main reader, then BufferedReader's readLine () method will return zero. Read the characters from BufferedReader Read () the Java BufferedReader method returns int, which contains the meaning of the symbol of the next character to read. If the method read returns -1, there is no more reading data in BufferedReader and it may be closed. That is, -1 as an int value, not -1 as byte or char value. There is a difference here! Here's an example of reading all the characters from Java BufferedReader: Reader - новый BufferedReader (новый FileReader(/path/to/file/thefile.txt);; int theCharNum theCharNum while (TheCharNum! - 1) - char theChar (char) theCharNum; System.out.print (TheChar); theCharNum - reader.read(); Notice how the code sample first reads one character from Java BufferedReader and checks whether the number is char -1. If not, it handles this character and continues to read until -1 is returned from the BufferedReader read method. As mentioned earlier, BufferedReader will actually read an array of characters from the main reader and return those characters one by one rather than renaming each () call to the main reader. When all the internal buffer characters have been read, BufferedReader tries to top up the buffer again until more characters can be read from the main reader. Read Array characters from the BufferedReader Java BufferedReader class also has a read () method that takes the array symbol as a parameter as well as the beginning of bias and length. An array of symbols is where the reading method will read the characters. The offset option is where the reading method should start reading in the char array. The length option is the number of characters that the reading method should count into an array of characters from the bias and forward. Here's an example of reading an array of characters in an array of characters from Java BufferedReader: Reader Reader - the new BufferedReader (/the new FileReader (/path/to/file/thefile.txt))) char'theChars - new char int charsRead - reader.read (theChars, 0, theChars.length); While (charsRead ! - 1) - System.out.println (new line (theChars, 0, charsRead)); charsRead - reader.read (theChars, 0, theChars.length); The reading method (enchantment, bias, length) returns the number of characters read in an array of characters, or -1, if BufferedReader no longer has characters, for example, if the end of the file to which BufferedReader is connected has been reached. Read a line from BufferedReader Java BufferedReader has a special reading method called readLine, which reads the full line of text from the internal BufferedReader buffer. The readLine method returns the line. If there are no more lines to read from BufferedReader, the readLine method returns zero. Here's an example of reading lines of text file one by one using Java BufferedReader : BufferedReader bufferedReader - the new BufferedReader (/new FileReader (/path/to/file/thefile.txt))) Line line - bufferedReader.readLine While (line!' null) - System.out.println (line); line - bufferedReader.readLine Reading the performance of Reading an array of characters at a time is faster than reading a single character at a time, from Java Reader. However, since BufferedReader already does some internal buffering, the difference is likely not as dramatic as with a reader that doesn't use buffering. You will most likely still see a small difference though. Skip the characters The BufferedReader class has a method called skip, which can be used to skip a number of characters in the input that you don't want to read. You pass the number of characters to skip as a parameter for the skip () method. Here's an example of missing characters from Java BufferedReader: long charsSkipped - bufferedReader.skip (24); This example tells Java BufferedReader to skip the next 24 characters in BufferedReader. The skip method returns the actual number of characters that have been missed. In most cases, it will be the same number as the one you missed, but if BufferedReader has fewer characters than the number of missed requests, the number of characters you missed can be less than the number of characters you requested. Closing BufferedReader When you're done reading characters from BufferedReader you have to remember to close it. The closure of BufferedReader will also close a copy of Reader, from which BufferedReader reads. Closing BufferedReader is done by calling it close () method. Here's what the closure of BufferedReader looks like You can also use the try-with-resources design in Java 7. Here's how to use and close BufferedReader looks with an attempt with resources to build: Reader Reader and the new FileReader (data/data.bin); try (BufferedReader buffered Reader - new BufferedReader (reader) String line - bufferedReader.readLine (); While (line ! Please note that there is no longer an explicit close call of the method. Of which it reads, so the FileReader copy will be closed when BufferedReader is closed. that you create a new object, and more importantly, a new array of characters that is used as a buffer inside BufferedReader. This can put pressure on a Java garbage collector if the number of files or threads read is high, and if they are quickly read by each other. The alternative is to create a reusable BufferedReader where you can replace the main Reader source, so bufferedReader and its internal byte array buffer can be reused. To save you from I created such a reusableBufferedReader, and included code for further on this tutorial. First, I want to show you how using this ReusableBufferedReader looks. Create a reusableBufferedReader First you need to create a reusableBufferedReader. Here's an example of how to create a reusableBufferedReader: ReusableBufferedReader - a new reusableBufferedReader (new char-1024) ; This example creates a reusable BufferedReader with an array of 2MB characters (1024 and 1024 characters, 1 character and 2 bytes) as an internal buffer. Install the source When you created ReusableBufferedReader you need to install a reader on it to use as your primary data source. Here's how you install the Reader source on ReusableBufferedReader : FileReader reader - new FileReader (/mydata/somefile.txt); reusableBufferedReader.setSource:supplied SetSource () method actually returns a link to ReusableBufferedReader, so you can create a reusableBufferedReader and install a source in one instruction: ReBufferedReader reBufferedReader - new reusableBufferedReader (new byte 1024) .setSource (new FileReader). Reuse reusableBufferedReader When you're done using ReusableBufferedReader you need to close it. Closing it will close only the main source of Reader. After closing ReusableBufferedReader you can use it again just by installing a new Reader source on it. here's how it looks to reuse ReusableBufferedReader: reusableBufferedReader.setSource (/mydata/file-1.txt)); Read data from ReusableBufferedReader reusableBufferedReader.close reusableBufferedReader.setSource:supplied by FileReader (/mydata/file-1.txt)); Read data from ReusableBufferedReader reusableBufferedReader.close Reusable FreshReader Here is the code for reusableBufferedReader, described above. Note that this implementation only overlaps the reader's reading and reading methods (char'dest, int offset, int length) of the Reader class that it expands. The rest of Reader's methods have been left to keep the code shorter - but you can implement them yourself in case you need them. import java.io.IOException Import java.io.Reader Public Class ReusableBufferedReader Expands Reader - Private Char' Buffer - null; Private Int writeIndex No 0; private int readIndex No 0; private boolean endOfReaderReached - false; Private reading source - zero; public reusable Bufferreader (buffer) - this.buffer - buffer; this.endOfReaderReached - false; Bring it back. - @Override public int read () throws IOException if (endOfReaderReached) - return -1; buffer.length) - this.writeIndex No 0; this.readIndex No 0; data should be read into the buffer. int bytesRead - readCharsIntoBuffer (); While (bytesRead No 0) actually get some bytes! by reutersRead and readCharsIntoBuffer (); If more data can't be read, return -1; If (bytesRead - -1) - return -1; Return 65535 - this.buffer (readIndex); - @Override int read (char'dest, int offset, int length) throws IOException - Int charsRead No 0; data No 0 int; While (data ! - 1 -1 - charsRead qlt; length); Data - read (); If (data -1) - endOfReaderReached - true; If (charsRead - 0); return charsRead; charsRead; Return charsRead; - private int readCharsIntoBuffer () throws IOException - int charsRead - this.source.read (this.buffer, this.writeIndex, this.buffer.length - this.writeIndex); writeIndex -charsRead @Override; }