Writing network code:
Handling TCP data in Streams
Crucial observations:
1. TCP transports 8-bit bytes (called OCTETs by the RFCs). This means that byte arrays are often a good representations of TCP data.
2. A TCP connection can close at any time. Thus, a write() operation must always be prepared to handle an IOException, and a read() operation must always be prepared to handle an IOException or an EOF.
3. Some parts of a message may be binary, while other parts may be just text (more specifically as 2822 section 2.2 says, headers fields contain characters in the US-ASCII range of 1 through 127).
4. TCP can break up the byte stream into segments at any point. A read operation can return any prefix of the bytes that have already been sent but not passed to the user. More simply, the last byte in a read operation might be anywhere in the byte stream.

What is a good way to read HTTP data from the network? What are the requirements? Overall:
• read bytes from the inputStream
• handle an IOException or an EOF at any time
• make sure we get the right characters
• make sure enough data has been read

There are two cases, with some additional (or more specific) requirements, reading headers or body:
1. reading headers
   1. obtain lines terminated by CRLF
   2. reject header fields containing the values 0 or in the range 128 to 255

reading a body
how do is the end determined?

To be completed ...
What are the tradeoffs of various streams that can be connected to a socket? Consider input streams first.

<table>
<thead>
<tr>
<th>Stream</th>
<th>Created</th>
<th>Advantages</th>
<th>Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputStream</td>
<td>java.net.Socket.getInputStream()</td>
<td>A non-abstract instantiation of InputStream which provides direct access to its methods and to bytes read from the socket.</td>
<td>Just an abstract class</td>
</tr>
<tr>
<td>FilterInputStream</td>
<td>FilterInputStream(InputStream in)</td>
<td>Buffers data read from the InputStream – in this case a socket – so that reads from the network obtain more bytes, thereby improving performance. Also, read() blocks.</td>
<td></td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>BufferedInputStream(InputStream in)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ByteArrayOutputStream</td>
<td>?</td>
<td>read() does not block.</td>
<td></td>
</tr>
<tr>
<td>DataInputStream</td>
<td>DataInputStream(InputStream in)</td>
<td>Reads java primitive types from the InputStream.</td>
<td>Not useful for protocols like HTTP and SMTP that send 2822 style messages</td>
</tr>
<tr>
<td>GZIPInputStream</td>
<td>GZIPInputStream(InputStream in)</td>
<td>Reads compressed data in the GZIP format.</td>
<td></td>
</tr>
<tr>
<td>InputStreamReader</td>
<td>InputStreamReader(InputStream in)</td>
<td>Converts bytes to characters.</td>
<td>If the charset is not specified, then the default can change the interpretation of bytes.</td>
</tr>
<tr>
<td>BufferedReader</td>
<td>BufferedReader(Reader in)</td>
<td>Provides readLine(), which MIGHT be helpful in some circumstances.</td>
<td></td>
</tr>
</tbody>
</table>

**readLine()**

Internet and Intranet Applications and Protocols
Spring 2005
Prof. A. Goldberg

In general java.io.DataInputStream.readLine() and BufferedReader.readLine() do not work because readLine() returns whenever it finds CR, LF or CRLF. They fail in several ways:

1. If the data is binary, such as gzipped data, any pattern can arrive, including CR, LF or CRLF. Since readline() does not indicate what terminated the line, it is impossible to reconstruct the data sent by TCP.
2. Suppose readline() is used to read headers. That should work OK shouldn't it, because each header line is terminated by CRLF, right? Well actually no. Two problems arise:
   a. If the header is terminated by just CR or LF it would be nice to catch and report the error, but readline() makes this impossible.
b. Suppose the sender *is* terminating lines with CRLF. Then, timing can cause the following behavior:
   i. TCP buffers a header line terminated by CR; readline() returns the line.
   ii. TCP now buffers LF ... ; readline() returns a line that it should not return!