Some grading remarks for Assignment 1 are as follows. The headers represent the grading remarks used, while the text that follows it further explains what it means.

No pipelining for HTTP 1.1; only one request is handled per connection.

This refers to the case where your program only handles one request for HTTP 1.1, and then handles no further responses. This grading remark usually means that either your handleClientSocket method does not have a loop in its body (thereby preventing it from ever handling more than one request), or does not use recursion (although recursion has problems of its own, described later).

HTTP 1.1 reads in only one request due to the presence of multiple buffered readers/scanners for subsequent requests.

This refers to the case where your program should in theory be able to handle multiple requests (since it loops and/or uses recursion), but for some reason doesn't. As buffered readers/scanners will attempt to greedily read in more input than necessary for a read() call (eg. reading in several lines of input when using a readLine() or a nextLine()), and store it in a buffer to make future read() calls faster, this can lead to the first buffered reader/scanner reading in the entire input stream of a socket. Once the next buffered reader/scanner is created, it attempts to read from the same input stream, but is now unable to do so, as no input can be found on the stream. You may wish to try the sample programs found in the BufferedDemo folder to see this behaviour in action.

As mentioned earlier, recursion in this assignment tends to have problems of its own, as students attempting to use recursion will simply pass the client socket into the parameter of handleClientSocket, and then create a new buffered reader/scanner in the next call. Due to this, programs using recursion are more prone to making this mistake as compared to iterative loops.

Note: this is the most common mistake made in this assignment.

HTTP 1.1 connection not properly closed after a timeout.

This refers to one of several possible cases:

1. Your program has no timeout implemented.
2. Your program attempts to implement a custom timeout (the most common form of this is having while loops, where the loop condition is that it has been less than X seconds since the socket was opened), which does not timeout properly.

3. Your program uses the setSoTimeout() method of Java's Socket class, but doesn't actually have a chance to trigger the timeout (the API documentation states the timeout only triggers upon a read() method on the underlying input stream).
4. Your program uses the setSoTimeout() method, properly timeouts via a read() method, but doesn't close the socket upon timeout (the API documentation states the socket is still valid even after the SocketTimeoutException has been thrown).

HTTP 1.0 handles multiple requests on the same connection.

Pretty much the opposite of not implementing pipelining in HTTP 1.1, in this case, multiple requests are handled on HTTP 1.0. Some possible reasons are:

1. Not distinguishing between HTTP 1.0 and HTTP 1.1, and implementing pipelining behaviour as the default.
2. Reading in all input from the stream first, sending response(s) based on the request(s) found in the read input, before deciding to close the socket (if on HTTP 1.0) or keep it open (if on HTTP 1.1).

HTTP 1.0 does not close the connection after a response.

In this case, after sending a response through HTTP 1.0, the socket is not closed. Unlike with *curl*, which can determine the end of the response via comparing the length output from your program with the Content-Length header, and then closing automatically once the required number of bytes has been received, other programs such as *telnet* or *netcat* cannot determine the end of the message automatically, and will keep the connection open if your program does not close the connection by itself.

Able to handle multiple files via HTTP 1.0.

Not exactly a mistake, but in this case, your program was unable to support downloading multiple files via HTTP 1.1 (which is what the provided script uses). However, your program is able to support downloading multiple files using HTTP 1.0, and was able to pass the grading rubric for handling a web page with multiple static objects.