**Objective**  To gain experience with Streams.

This week's assignment will give you the opportunity to practice working with Streams by writing the networking part of last week's "Spam" program. This week you will complete the `MailConnection` class, which was provided to you in a jar file last week. The class implements the mail protocol by opening a socket and input and output streams, writing messages to the server, and interpreting the responses.

Before getting into the details of `MailConnection`, let us quickly review of the mail protocol. What follows is an example session involving connecting to a mail server.

telnet mailhost.cs.williams.edu 110
Trying 137.165.8.2...
Connected to bull.cs.williams.edu.
Escape character is `^]`.
+OK POP3 bull v2001.78 server ready (*)
USER test4
+OK User name accepted, password please
PASS test
+OK Mailbox open, 157 messages
STAT
+OK 157 3883728
TOP 154 0
+OK Top of message follows
Return-Path: <jnfoster@seas.upenn.edu>
Received: from lion.seas.upenn.edu (LION.SEAS.UPENN.EDU [158.130.12.194])
    by bull.cs.williams.edu (8.12.3p3/8.12.3) with ESMTP id hAKFDkNF016729
    for <kim@cs.williams.edu>; Thu, 20 Nov 2003 10:13:46 -0500 (EST)
    (envelope-from jnfoster@seas.upenn.edu)
...(other received info omitted)
Date: Thu, 20 Nov 2003 10:13:45 -0500 (EST)
From: Nate Foster <jnfoster@seas.upenn.edu>
To: Kim Bruce <kim@cs.williams.edu>
Subject: Re: [cvs-loojc] loojc/ecoop main.ps main.tex
In-Reply-To: <DC018B3E-1B06-11D8-BC26-000A959A07CA@cs.williams.edu>
Message-ID: <Pine.GSO.4.58.0311201003320.10597@blue.seas.upenn.edu>
References: <Pine.GSO.4.58.0311191311270.18338@blue.seas.upenn.edu>
    <F44C043D-1AC7-11D8-BC26-000A959A07CA@cs.williams.edu>
    <Pine.GSO.4.58.0311191459190.5095@blue.seas.upenn.edu>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Status: RO
.
QUIT
+OK Sayonara
Connection closed by foreign host.
Your MailConnection class will need to encode most of this protocol. We will provide you with our solution to last week's SpamFilter class to use with your new MailConnection class.

See the handouts web page to obtain a copy of the starter code.

---

**The Design**

The MailConnection class manages the communication with a mail server. We begin with the descriptions of the private methods we have provided in the class for your use:

- `private void netPrintln(String s)` Writes a string to the mail server's output stream.
- `private String netReadLine()` Read a line from the mail server.
- `private void close()` Closes the socket. Must be called if the connection was not established in the constructor.

*Note: We have included a constant, DEBUG that is initially set to be true. It causes anything written to the mail server by netPrintln to be echoed to the console window in Eclipse. Similarly anything read from the mail server by netReadLine is also echoed to the console window. It should be set to false before you turn in your program.*

We also provide two public methods: disconnect, which closes the connection after a successful mail session, and isConnected, which returns a boolean indicating whether the connection is currently active with the mail server. These are the methods that you called last week from your SpamFilter class.

The following is a description of the constructor and methods that you must complete.

- `public MailConnection (String host, String userName, String password)` Create a mail connection to a host for a specific user. If everything works, set connect to true and return. If it fails for any reason, pop up a dialog box with a helpful message and call the close() method to close the connection.

  We have provided the code to create a new socket as well as input and output streams (called `input` and `output`) that use that socket. You should read the response from the mail server (using the private `netReadLine()` method) starting with the line marked (*) in the sample above. If that line starts with "+OK" then go through the protocol identifying the USER and password. If all of the responses start with "+OK" then set boolean variable connected to true. Otherwise pop up a dialog box providing information on what went wrong (see the provided code).

- `public int getNumMsgs()` Returns the number of messages in the mailbox you are connected to. This returns 0 if there is no active connection.

  This method will send a STAT command to the server and extract the number of messages available from the line the server sends back. It will then return the integer obtained from that string. Recall that the method `Integer.parseInt(s)` converts the digits in the `String s` to an integer and returns that value. For example, `Integer.parseInt('33')` returns the value 33.

- `public String header (int msg)` Returns the headers of a mail message identified by the number passed in. Unlike Java, mailboxes number messages beginning with 1 and go up to the number of messages contained in the mailbox. To obtain the header for message i, you must send “TOP i 0” to the mail server.

  header

  returns an empty string if it is called when there is no connection or if the `msg` parameter is less than 1 or greater than the actual number of messages.

  This method should use `readResponse()` (see below) to obtain the String consisting of the full header.
private String readResponse() {
    Reads a multi-line response from the mail server. Returns an empty string if there is no connection.

    If the connection is alive then it should successively read new lines from the input stream and concatenate them together (inserting new line characters at the end of lines) until the results of a read are either null or ".".

    Be sure to test the constructor and each method you are writing carefully before going on to the next one. To make this easy, the version of our solution to last week’s lab which we have included does not perform the entire process of fetching and filtering the mail as soon as you click one button. Instead, the first time you click the button it merely connects to the server. The next time you click, it gets the number of available messages. After that, each message downloads one message until they have all been fetched. Once your program is working, you can change our program to behave more like last week’s lab by removing one line in the begin method as indicated by the comments in our code.

--- Submitting Your Work ---

When your work is complete you should deposit in the appropriate dropoff folder a copy of the entire MailStreamStarter folder. Before you do this, make sure the folder name includes the phrase “Lab 11” and your name. Also make sure to double check your work for correctness, organization and style. This assignment is due at the end of lab today.

--- Grading Guidelines ---

<table>
<thead>
<tr>
<th>Value</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Syntax Style (4 pts total)</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Descriptive comments</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good names</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Good use of constants</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Appropriate formatting</td>
</tr>
<tr>
<td></td>
<td>Semantic style (2 pts total)</td>
</tr>
<tr>
<td>2 pt.</td>
<td>General correctness/design/efficiency issues</td>
</tr>
<tr>
<td></td>
<td>Correctness (4 pts total)</td>
</tr>
<tr>
<td>1 pt.</td>
<td>Constructor</td>
</tr>
<tr>
<td>1 pt.</td>
<td>getNumMessages</td>
</tr>
<tr>
<td>1 pt.</td>
<td>header</td>
</tr>
<tr>
<td>1 pt.</td>
<td>readResponse</td>
</tr>
</tbody>
</table>