CS 360 Spring 2012 PUIM Command Line Client and Library **Assignment Two Point Five**
DUE: March 21, 2012, 2:15 pm **50 points**

For this project, edit your Command Line client to reflect protocol changes. Fix your bugs!

The PUIM (Pacific University Instant Message) has changed!

PUIM Version 2
The protocol messages are specified below.

Each line of text is a maximum of 1024 characters including the \n.
Each line must be terminated by a \n.
Lines marked with C are produced by the client. Lines marked with S are produced by the server. Neither C nor S are transmitted.

For ERR messages, any string after the ERR is optional.

Session Startup.
C HELLO username
S OK | ERR Username already in use

Session Disconnect
C DISCONNECT
S OK | MSG FROM:username

Send a message to another user.
C MSG TO:username
S OK | ERR Username not found | MSG FROM:username
C <TEXT terminated with single . on a line>
C .
S OK

Receive a message from another user.
S MSG FROM:username
C OK
S <TEXT terminated with single . on a line>
S .
C OK

CONFLICTS

If a server and client both initiate the send of a message (text, data) the CLIENT must back off and accept the server's message by sending an OK. The client then attempts to resend its message after the server completes its transaction.

If the client requests to DISCONNECT but receives [MSG|DATA] FROM:username rather than OK, the client must receive the full message and after the final OK then try again to DISCONNECT. Once a

server has received DISCONNECT (even if that DISCONNECT is interrupted by a message from the server) no new messages [MSG|DATA] can be sent to the client.

Example:

```
C MSG TO:doug
S MSG FROM:shereen
C OK
S <TEXT terminated with single . on a line>
S .
C OK
C MSG TO:doug
S OK
C <TEXT terminated with single . on a line>
C .
S OK
```

ADDITIONS

If any line of data is too long the server will send the message:
ERR too long
and terminate the current transaction (stop receiving the message or data).

Data messages

```
C DATA TO:username
S OK | ERR Username not found | ERR unsupported
C CONTENT/TYPE: mime/type
C <TEXT terminated with single . on a line> (base64 encoded image)
C .
S OK

S DATA FROM:username
C OK | ERR Username not found | ERR unsupported
S CONTENT/TYPE: mime/type
S <TEXT terminated with single . on a line> (base64 encoded image)
S .
C OK
```

Update to TEXT

Using a single period on a line as a message termiantor prevents the user from sending a single period on a line as part of a message. This is unacceptable. To remedy this, we will adapt the procedure outline in the SMTP RFC:
http://tools.ietf.org/html/rfc5321#section-4.5.2

- Before sending a line of **text**, the client checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.

- Before displaying a line of **text**, the client checks the first character of the line. If that character is a period, the first character of the line is deleted.

- Note that the terminator (.\n) is not part of the **text** and so does not change.

- The server has no role in this change.

The Command Line Client

Fix your bugs in the command line client.

The command line client must take three command line options: server address, server port, and username. The user can type three commands. EXIT cleanly terminates the connection and exits the client. SEND username message, will send a message to user username. DATA username filename, will send the file, filename to the user username, base64 encoded. While the user is typing at the console, any incoming messages must also be displayed.

To read a file from the Eclipse root directory, or the current working directory, use **./filename**

bart$ java -jar PUIMCommandLineClient.jar coffee.cs.pacificu.edu 12349 chadd

> SEND chadd This is the message
chadd >> This is the message
> DATA chadd ./boxer.png
chadd >> FILE tmp123.png received.
> EXIT

bart$

Server:
A simple server will be provided on Monday. You can run this server as follows:

bart$ java -jar PUIMSimpleServer.jar 12349

Eclipse Project

No new Eclipse project should be produced.

Base 64 encoding: You must download and install into your Eclipse project the Apache Commons Codec library (binaries, version 1.6).

Download:	http://commons.apache.org/codec/download_codec.cgi
Documentation:	http://commons.apache.org/codec/api-release/index.html

Create a **lib** directory in the appropriate Eclipse project and unzip the downloaded code into that directory. You should see a number of jar files (after an F5 refresh). Add those jar files to the Java Build Path for the project.

Right Click project, Build Path, Configure Build Path, Libraries, Add JARs, navigate

You can use **String Base64.encodeBase64String(byte [])** and **byte[] Base64.decodeBase64(String)**, to encode and decode data, respectively. You will be reading a file into a **byte []**, and sending a **String** across the network. To access a file use **FileOutputStream** and **FileInputStream**, and **File** to get the length of the file. **Base64InputStream** and **Base64OutputStream** may also work.

When a user sends a file, generate the CONTENT-TYPE based on the file extension. When a user receives a file, generate a random file name and use the CONTENT-TYPE as the file extension. Display a message to the user detailing what file was received and from who (as shown above). **File.createTempFile(String, String, File)** may be helpful.

The 1024 byte line limit still applies for DATA.

Be aware that this library may insert a \n after every 76 characters of base64-encoded data.