Roll your own AIM Client!

For the rest of this week, we're going to be working on aspects of an AOL Instant Messager client. By the end of the week, you should be able to exchange IMs with your friends using a client that you wrote yourself!

Overview

AIM uses two protocols: OSCAR and TOC. OSCAR is the protocol that most AIM clients use, be it AOL's own client, GAIM, or Trillian. Unfortunately, OSCAR is a binary protocol—so it requires manipulating bytes rather than Strings in order to do anything—and, even worse, undocumented. TOC, on the other hand has fewer features, but is text based and documented. We're going to use TOC and have provided the documentation for it on our homepage.

As a historical note, there are two versions of TOC: TOC and TOC2. TOC is no longer supported by AOL and no AOL servers recognize it. We're going to be using TOC2, which fixes some of the bugs in the TOC protocol and adds new features.

The TOC protocol is pretty simple. After an initial song and dance to get a connection, you send text messages to the server when you want to send an IM and the server sends text messages back to you. If there is an error, the server will either send you an error message or drop the connection. (This is a simple behavior, but pretty annoying to debug.) To do simple IMMing you only need to support the following messages.

- (Server to Client): SIGN_ON, IM_IN2, ERROR
- (Client to Server): toc_signon, toc_init_done, toc2_send_im

The TOC messages themselves are wrapped in FLAP packets. We suggest that you read through all of the TOC documentation to get an idea what is going on with the protocol and exactly what each of the protocol messages do.

Support Code

We've provided a bunch of support code and for building your AIM client. The functionality is broken into three main layers, with each layer represented by a Java class and a set of `TOCMessage` sub-classes that provide Java object representations of TOC protocol messages.

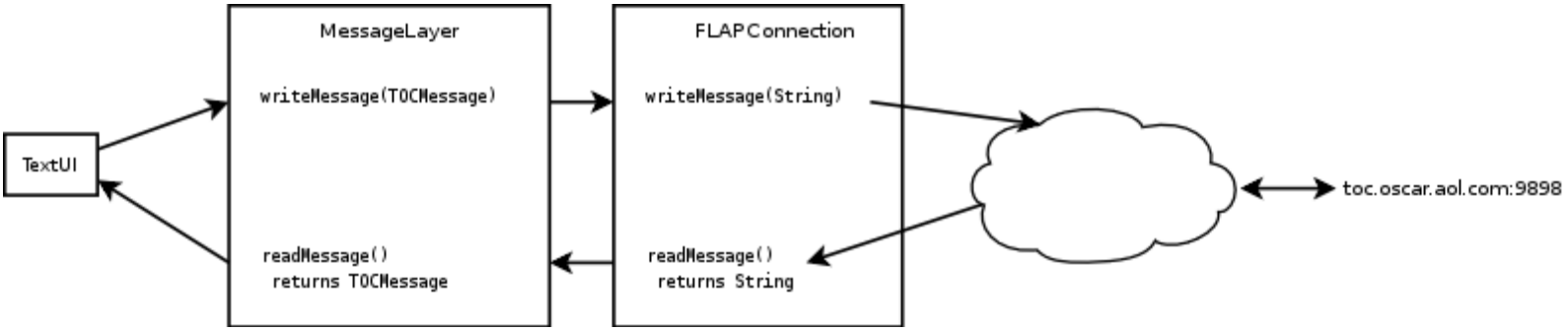
- `edu.mit.tbp.se.chat.ui.TextUI` — a simple text-based user-front end. The one main command it provides is `sendim buddyname message...`. Try IMming people once you get the `>>>` prompt.

  The `TextUI` class communicates with lower-level classes using `TOCMessage` objects. Each message to or from the server is represented as a `TOCMessage`.

- `edu.mit.tbp.se.chat.message.MessageLayer` — lower-level code that deals with sending and receiving TOC protocol messages to and from the server.

  The `MessageLayer` class relies on the individual `TOCMessage` sub-classes to convert themselves into the protocol strings that the TOC protocol expects. You're going to spend most of your time today writing code at this level.

- `edu.mit.tbp.se.chat.connection.FLAPConnection` — very low level code that deals with sending and receiving bytes from the server using the FLAP framing protocol.



As the figure above illustrates, there are two main paths through these classes. The message sending path starts when the `TextUI` calls `sendMessage()` in the `MessageLayer`. `MessageLayer.sendMessage()` in turn turns the `TOCMessage` into a string and calls on `FLAPConnection.writeMessage()`. `FLAPConnection.writeMessage()` finally converts the TOC protocol string into a bunch of bytes and sends them across the internet to the TOC server.

The message receive path also starts at the `TextUI`. To get the next message, the `TextUI` calls `MessageLayer.receiveMessage()`, which calls `FLAPConnection.readMessage()`. `FLAPConnection.readMessage()` waits until it has a complete message from the TOC server. When it does, `readMessage()` returns it as a string, which the `MessageLayer` turns into a `TOCMessage` , which finally gets delivered to the `TextUI` The `MessageLayer` class converts Strings into `TOCMessage` objects that the user interface can use.

We've provided, on our homepage, a jar of our implementation so you can get a feel of what we're going to be doing before you write yours. To run our implementation, download our jar file and type:

```
java -jar saim.jar username password
```

to start the client. You should get a command prompt. Type `help` to see what options are available to you. If start the client with no options, it will give you a command-line usage statement. It's probably useful to play with the `--log` option. Our code outputs protocol messages at logging level FINER and byte-level output at logging level FINEST.

Assignment

We're going to implement the `MessageLayer` class and a few `TOCMessage`s to make using the class easier. Before you get started, read the TOC protocol documentation and read through the documentation for our classes on the website. In particular, pay attention to the specification for each method and the different kinds and try to relate their function to parts of the TOC protocol.

There are three main methods that you need to implement in the `MessageLayer`: `login`, `sendMessage`, and `receiveMessage`. `login` should use the other two methods, so it's probably a good idea to implement them first and test them using the `login` method.

You'll also need to implement a `TOCMessage` sub-class for each kind of message you'll need to send—more or less the list we presented in the TOC introduction. To support sign-in, you need to be able to respond to the SIGN_ON message as well as send the toc_signon, and toc_init_done messages. To support really basic IMming, you need to be able to send toc2_send_im messages and receive IM_IN2 messages. The `TextUI` knows how to handle the `ServerIMIn2Message` and `Toc2SendIMMessage` classes, but you must provide implementations for them. Finally, we've provided code that handles the ERROR message from the server and turns them into checked exceptions.

We've provided specifications and starter code for most of the classes you need to implement. Feel free to eschew our system entirely, but you'll probably get further working with our code.

Help

Don't forget to ask for help. This is a non-trivial assignment and we're here to help you get through it.

Logging

Our code uses the java.util.logging API to control how much debugging output you get. If you give our code a `--log=LEVEL` on the

command line, you can get more debugging information. To see what protocol-level options are being sent, set it to the FINER level.

To use the logger in your code, read the Logger class's API and mirror the usage in our code.

More Stuff

If you get your client working, there's more you can do. If you want to do more low-level code, try your hand at implementing the `FLAPConnection` class. In doing so, you will learn how to use network sockets and the joy of manipulating byte streams to do your bidding. If that's not your cup of tea, try building your own UI, maybe using a GUI. Finally, TOC2 has a bunch of useful commands for manipulating buddy lists and block lists, so an easy extension might be to add buddy list capabilities.

In any case, find a staff member and go over what you want to do so we can help.