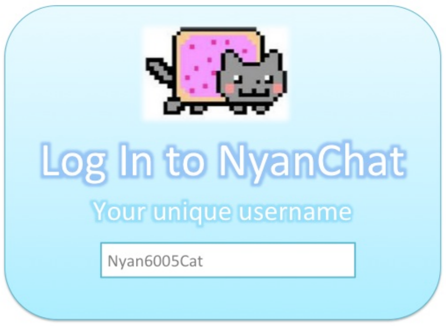
DESIGN DOCUMENT
6.005 Project 2: “NyanChat”
Will Grathwohl, Jesika Haria, Anvisha Pai

[What inspired NyanChat? http://en.wikipedia.org/wiki/Nyan_Cat]

1. Conversation design

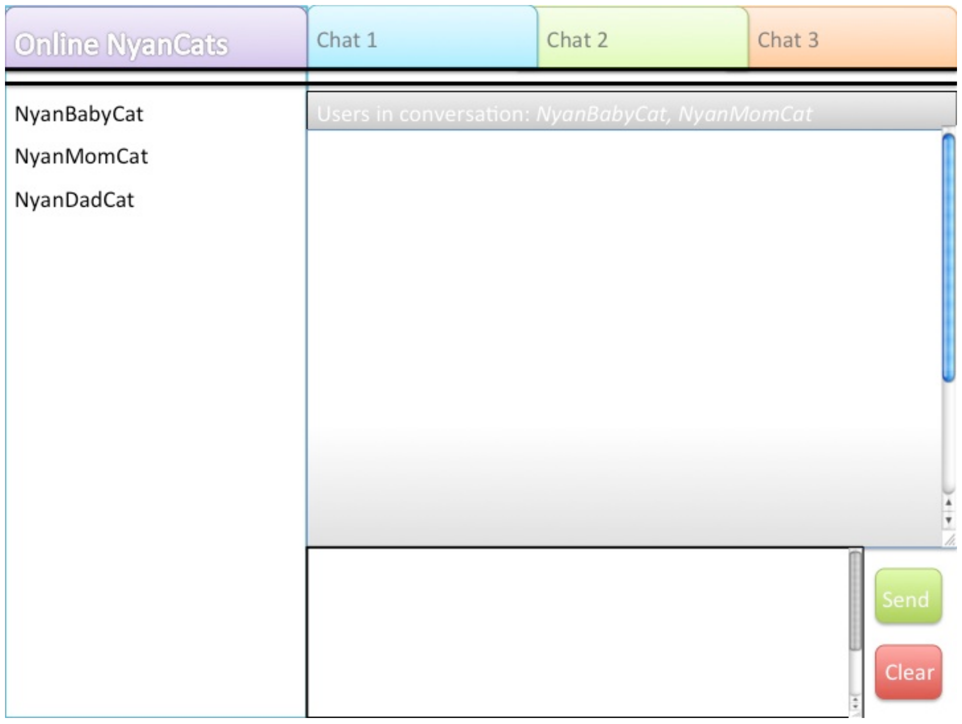
1.1 High-Level Summary:

The user signs in with a username. When they hit “sign in”, they are directed to a window that looks like this:



The user can only chat in one conversation at an instant (this is enforced by the use of tabs).

They can start a new conversation with another online user by clicking on their name in the Users Online panel. New users can be added to the conversation. Theoretically, a user can have multiple conversations with another user (each conversation has a unique conversation ID). As of now, users cannot deny a conversation. Conversations do have a name (or a default name Chat x). According to our definition, conversations cannot be empty.



1.2 Classes:

1.2.0 MAIN

Server:

Main method: calls ChatServer constructor

Client:

Main method: calls ChatGUI and ChatClient constructors

1.2.1 MODEL

Server-Side

To define a conversation we define three main server-side classes.

ClientHandler:

-Handles connection from the server to one client. Listens to messages sent from the client. Contains pointer to main server object.

-Has methods:

SendToClient(String message): Sends messages to the ClientHandler's client. Message is the message to be sent to the client, determined by the server.

SendToServer(String message): Relays message from the client to the main server object where it will be parsed and handled.

ConversationHandler:

-Represents each current conversation. Contains ArrayList<ClientHandler> for each user in the conversation. Also contains a String with the conversation ID.

-Has methods:

AddUser(String username): Adds a user to the conversation. Will be called by the server. username will come from a client message.

RemoveUser(String username): Removes a user from the conversation. username will come from a client message.

SendMessage(String message): Sends text to each user in the conversation. message will come from a client message.

ChatServer:

-Is the main server object. Handles all messages sent from the clients. Sends messages to ClientHandlers or to ConversationHandlers depending on the messages it receives.

-Has methods:

ChatServer(int port): Called from main method in Server class

HandleMessage(String message): Parses message, creates ConversationHandlers and ClientHandlers as needed, and passes messages to ClientHandlers and ConversationHandlers. message will come from a client, relayed by a ClientHandler.

<u>Client-Side</u>

ChatClient :

-extends Thread class that communicates with ClientHandler i.e. Sends messages to the ClientHandler/Server and receives messages.

-Has public methods:

ChatClient(String IPAddress): Called from the Client main method

void run(): writes from and reads to server.

when reading from the server (if not null) it creates a new Controller object

checks ChatGUI.getQueue (which returns a BlockingQueue) and sends that message to the server

<u>1.2.2 CONTROLLER</u>

Controller (extends SwingWorker):

Has **public** methods:

Controller(String message): called from ChatClient

(String) doInBackground(): returns message

done(): mutates GUI

<u>1.2.3 VIEW</u>

ChatGUI:

- this class implements Swing to create the GUI Frame. The Frame at first has a visibility set to False and opens up a dialog box which has the login screen.

-Has **public** methods:

ChatGUI(): Initializes components and BlockingQueue messageQueue, has actionListeners that update messageQueue

BlockingQueue getQueue(): returns messageQueue

1.2.4 HELPER METHODS/CLASSES

We are going to define an enum class with command types and a Parser class to parse commands, this will help with the processing in the main program body.

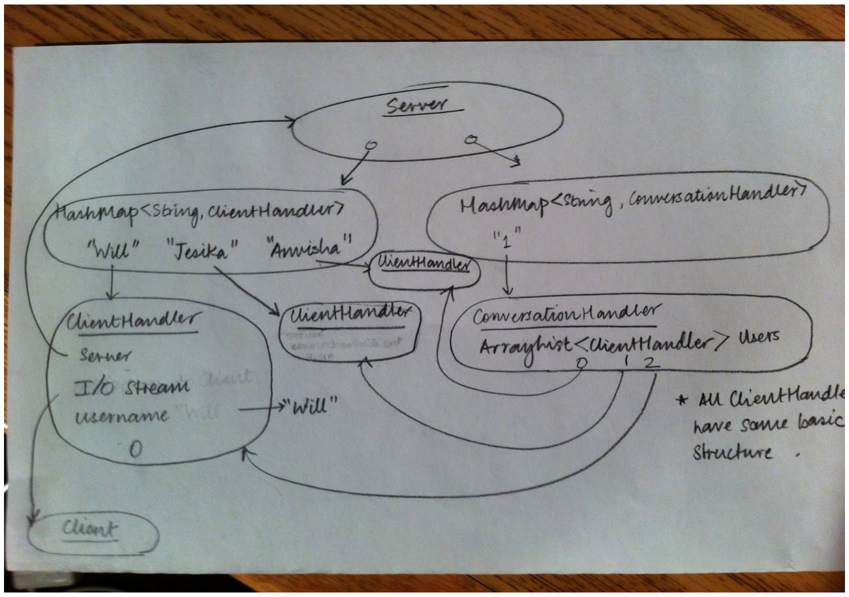
2. Protocol

Message ::= Client-to-Server | Server-to-Client
Client-to-Server ::= New-Conversation-Request | Add-User-Request | Chat-Message | Sign-In-Request
New-Conversation-Request::= “CREQ SENDER ” Username “RCPT” Username end-line
Add-User-Request::= “ADDUSER ” Username “CID” Conversation-ID end-line
Chat-Message ::= “CHATSENDER ” Username “CID” Conversation-ID “DATA” Data end-line
Server-to-Client ::= Chat-Message-or-Error | Conversation-Started | Conversation-Users-Change | Online-Users | Signed-in
Chat-Message-or-Error :: = (Chat-Message | “CHAT ERROR”) end-line
Conversation-Started ::= “CSTART ” (“CID” Conversation_ID “USER” Username “USER” Username | “ERROR”) end-line

Conversation-Users-Change ::= “CCHANGE ” (“CID” Conversation_ID (“ADDED” | “REMOVED”) Username)|“ERROR”) end-line

Online-Users ::= “ONLINEUSERS ”(Username)* end-line
Signed-In:: = (“SIGNEDIN ” Online-Users|”ERROR” (“Username in Use”|”Connection Error”))
Username ::= \w+
Conversation_ID ::= \d+
Data::= (\w|\s|\n|\r\n)+
end-line :: = \n | \r\n

Snapshot Diagram:


Server
HashMap<String, ClientHandler>
"Will"
"Jesika"
"Anvisha"
HashMap<String, ConversationHandler>
"1"
ClientHandler
ClientHandler
ClientHandler
Server
I/O Stream
username
"Will"
0
ConversationHandler
ArrayList<ClientHandler> Users
0
1
2
* All ClientHandle have same basic Structure.
Client

Message Trace-Through:

In the current system if user “Will” sends the message “Hey guys”, to the user “Anvisha,” a new conversation will be initiated The messageQueue for Will will have 2 items. Will’s ChatClient checks the messageQueue and sends the message:

“CREQ SENDER Will RCPT Anvisha”

to the server. The server then sends the message:

“CSTART CID 1 USER Will USER Anvisha”

to both Will’s client and Anvisha’s client. Will’s and Anvisha’s ChatClients then receive the message and create new Controller objects to update the GUI.

Will’s ChatClient then sends the next messageQueue item:

“CHATSENDER Will CID 1 DATA Hey guys”

to the server. The server then sends the data “Hey guys” to Server.ConversationHandlers[1] which sends the the data to each ClientHandler in Server.ConversationHandlers[1].ClientHandlers where each intended user will receive the message. Similarly, the ChatClients of each user will receive the message and create new Controller objects to update the GUIs.