6.005 Project 2: Conversation Design and Client-Server Protocol

Team Members: Rebecca Zhang (ryzhang), Katie Lee (katielee), Birkan Uzun (birkanu)

TA: Joshua Oreman

Conversation Design

Definition of Conversation: Communication of text between two or more clients on the server.

Our Instant Messaging System will allow a conversation between two or more people in the chatroom. There is one chatroom provided by the Server and users may come and go in the chatroom. All online users automatically join the chatroom and can see all other clients and their messages are displayed to all other users.

Overview of Public Classes

Chat Package:

- <class> Server
  - Handles connections for and communication between multiple clients
  - Variables:
    - users: HashMap that keeps track of current instances of User that are connected to the server, key: String username, value: User that is associated with that username
  - Methods:
    - serve(): creates new instance of User class as a thread when a new user connects with a username
    - handleRequest(String input): make requested mutations to the Chat class if applicable, then return appropriate message to the user (See Server-to-Client protocol)
- <class> clientThread
  - individual thread of the server that handles that specific client's requests
  - clientThread takes in the socket, port, and clientThreads list
  - thread responsible for client involved broadcasts to every other client
  - LoginWIndow pops up after user connects to Server's port.
  - clientThread is started before MainWindow

    - clientThread has its own username associated with it
- <class> Client
    - Creates a clientThread, but is NOT a thread.
    - Choose an individual user alias to take for the chat time.
    - works with publisher class to publish events to the GUI.
    - startDialog():
        - opens a new Socket and connects to the ServerSocket (given the host and port)
        - creates an output (ObjectOutputStream) to write to the Server and input (ObjectInputStream) to read from the Server
        - starts the Publisher which announces Events to the GUI
    - closeConnection():
        - closes the input, output, and clientSocket
    - updateUserList():
        - updates the online clients list
- <class> Publisher
    - Publishes communication to listeners (we will be using it to publish from Client to GUI)
    - addInputStream(BufferedReader inputStream):
        - Define the input stream from which the publisher reads its messages from
    - addListener(Listener listener):
        - adds listener to ArrayList of all the listeners of the Server
    - announce(String event):
        - broadcasts the event to all listeners of online Clients
    - run():
        - Listens for communication from inputStream (sees server communication to client)
        - Publishes inputStream messages to the listeners (sends messages from client to GUI)
- <class> Event
    - has enum Types {AddUser, RemoveUser, Message}
    - Constructor takes in a Type type and String message
    - getType(): returns the type
    - getMessage(): returns the message
    - StringToEvent(): turns a String into an Event
    - toString(): turns an Event into a String

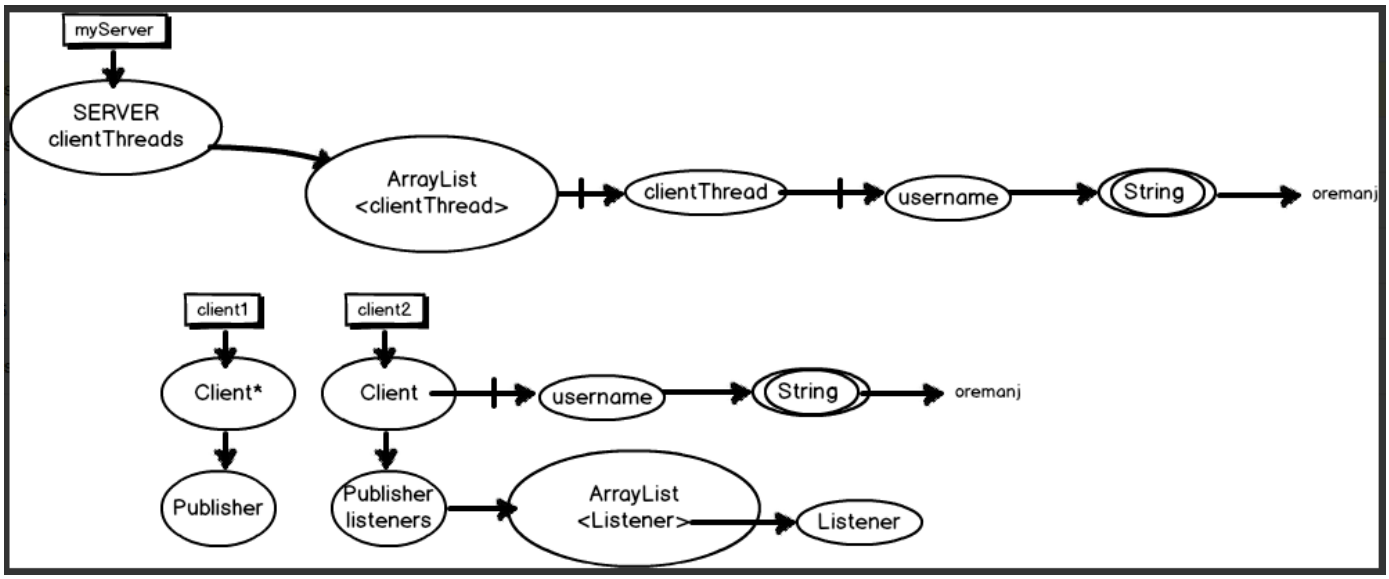
- <interface> Listener
    - implements event listener
    - required methods:
        - event(String serverResponse)

GUI Package:

- <class> LoginWindow
    - Created when the program begins
    - Instance of Client class is also initiated when the LogIn Window is created.
    - Client must login to specific instance of User with valid username and password to enter ChatWindow state
    - Components:
        - JLabels to label “GUIChat”, “Username:”, “host”, “port”
        - username (JTextField) → enter String username
        - host (JTextField) → enter host
        - port (JTextField) → enter port number
        - login (JButton) → if login information is correct, a Client is initiated, LoginWindow closes and FriendWindow opens. else, an error message pops up, and LoginWindow fields are cleared
            - we will make this login function threadsafe in the server so that two people trying to log in to the same User cannot both do so.
    - ActionListener
        - listens for Start Button to be pressed.
            - starts the MainWindow
            - closes own window
- <class> MainWindow
    - Window for a conversation between two or more users
    - Implements the Listener class in order to read events from the Client
    - Components:
        - newText (JTextPane) → enter text to be added to conversation
        - send (JButton) → send newText to conversation
        - conversationText (JEditorPane) → displays text exchanged in specific instance of GroupChatWindow’s conversation
        - participants (JList) → list of usernames of Clients participating in the group chat
    - ActionListener

        - listens for user to:
            - send Text - writes Event to the Server
            - disconnect - writes to Server saying this user has requested to log out
    - Event method from implementing the Listener class:
        - Parses messages from the Client using the Client:GUI grammar.

Snapshot Diagram



Timeline

1. Server starts with given host and port
2. When LoginWindow starts, user can type in host and port, and username, to join the chatroom.

3. When the submit button is pressed, an instance of Client is created that connects to the Server and starts listening to the Server's messages. LoginWindow disappears.
4. MainWindow is opened, which allows the user to start typing into the chatroom by pressing the "send" button.
5. User can disconnect by pressing the "disconnect" button.

5a. If so, then the Client relays the disconnect request to the Server and closes the connection.

Client-Server Protocol

The Client and Server classes need to communicate in order to log into a specific User and to send messages within a Chat. We have specified protocols to allow for the following:

- a client designating a username
- a client disconnecting from the server
- exchanging messages between users

Client-to-Server Protocol

MESSAGE :== (LOGIN | LOGOUT | SENDTEXT) NEWLINE

LOGIN :== "login" SPACE USERNAME

LOGOUT :== "logout"

SENDTEXT :== .+

USERNAME :== [^SPACE]{6,16}

SPACE :== \s

NEWLINE :== "\r?\n"

Server-to-Client Protocol

MESSAGE :== (BROADCAST_LOGGEDIN | BROADCAST_LOGGEDOUT | BROADCAST_MESSAGE | REMOVE_USER) NEWLINE

BROADCAST_LOGGEDIN :== USERNAME SPACE "has entered the chatroom."

BROADCAST_LOGGEDOUT :== USERNAME SPACE "has left the chatroom."

BROADCAST_MESSAGE :== USERNAME : SPACE .+

REMOVE_USER :== "RemoveUser" SPACE USERNAME

ALL_USERS :== “AllUsers” USERNAME (SPACE USERNAME)*

USERNAME :== [^SPACE]{6,16}

SPACE :== \s

NEWLINE :== "\r?\n"

Publish-Subscribe Protocol

(Client broadcasting messages to the GUI)

MESSAGE :== TYPE TEXT

TYPE :== “AddUser” | “RemoveUser” | “Message” | “AllUsers”

TEXT :== .+