User Experience

Login view presented on startup

- Text box: Server IP
- Text box: Port number
- Text box: Username entry
- Button: Login
    - Action: Checks for valid IP, port, username
        - If valid: Go to conversation view
        - Else: Error popup, returns to login screen.
- Color selection (default = black)

Conversation view

- Tabbed view
    - Tabs for each open conversation (on the left)
    - Frame containing active conversation's history
- Text field with send button for new messages
- New conversation starter
    - Select recipient from dropdown list of online users
- Buttons to close open conversations
- Logout button

Classes

User: Object representing user of a client

- `public User(String username, Color color, Socket socket);`
    - @param username, a String representing the User's name
    - @param color, a Color associated with the User in the UI
    - @param socket, the Socket associated with this User
    - `private ConcurrentHashMap<String, UserInfo> onlineUsers;`
        - Maps username strings to UserInfo object containing info about all other online users.
    - `private Conversation activeConvo;`
        - The current conversation the user has open.
    - `private ConcurrentHashMap<String, Conversation> myConvos;`
        - Maps convoID strings to Conversation object for all active conversations this User is part of.
    - `private ConcurrentHashMap<String, Color> colorMap;`
        - Maps String color names to actual Color objects
- `public void main() throws IOException;`
    - calls handleConnection
- `public void sendMessageToServer(String text);`

    - Sends text to the server, using this.socket.
- `public static void handleConnection() throws IOException;`
    - Handle connection to the server, using its this.socket
    - Calls handleServerRequest()
    - @throws IOException if connection has an error or terminates unexpectedly
- `public static String handleServerRequest(String input);`
    - handler for server messages. Update conversations, etc based on the message received.
    - @param input, the input from the server, from the grammar
    - @return the message (from the grammar) to the client
- `public String getUsername();`
    - @return String representing the User's username
- `public Color getColor();`
    - @return Color representing the User's color
- `public Socket getSocket();`
    - @return Socket representing the User's socket
- `public ConcurrentHashMap<String, UserInfo> getOnlineUsers();`
    - @return ConcurrentHashMap mapping usernames of all online users to their UserInfo objects
- `public void setActiveConvo(Conversation convo);`
    - updates the stored value for current conversation
    - @param convo, the new active conversation
- `public void startConvo(Conversation convo);`
    - call sendMessageToServer("-s" + convo.convo_id)
- `public void closeConvo(Conversation convo);`
    - call sendMessageToServer("-x" + convo.convo_id)
- `public void addMsgToConvo(Conversation convo, String text);`
    - call sendMessageToServer on the text from the message, the convo_id, and the username, according to the grammar.
- `public void quit();`
    - call sendMessageToServer("-q" + this.username) and closes its socket's connection.
- `public void setOnlineUsers(ConcurrentHashMap<String,UserInfo> userMap);`
    - Replaces the onlineUsers ConcurrentHashMap with userMap.
- `public void addOnlineUser(UserInfo user);`
    - Adds user to the onlineUsers ConcurrentHashMap.

- `public void removeOnlineUser(UserInfo user);`
    - Removes user from the onlineUsers ConcurrentHashMap.

Message: Object representing an instant message

- `public Message(UserInfo sender, Conversation convo, String text);`
    - @param sender, the UserInfo of who sent the message
    - @param convo, the conversation the message is part of
    - @param text, the body of the instant message

Conversation: Object representing a Conversation

- `public Conversation(ConcurrentHashMap<String, UserInfo> participants);`
    - @param participants, ConcurrentHashMap mapping usernames to UserInfo for all Users who are participating in the conversation
    - `String convoID;`
        - Identifier for conversation
        - Format: usernames of conversation participants in alphabetical order, separated by spaces.
    - `ConcurrentHashMap<String, UserInfo> participants;`
        - Username mapping to UserInfo for everyone participating in the Conversation
    - `ArrayList<Message> history;`
        - ArrayList of all messages in the conversation
- `public void addMessage(Message message);`
    - @param message, new message that will be added to the history.

ChatServer:

- `public ChatServer(int port) throws IOException;`
    - `ConcurrentHashMap<String, UserInfo> infoMap;`
        - maps String usernames to UserInfo objects which have a record of the user's relevant information (name, color, socket used for communication)
        - uses threadsafe implementation of HashMap
- `public void serve() throws IOException;`
    - Run the server, listening for client connections and handling them. Never returns unless an exception is thrown.

    - Uses a ServerThread class that extends runnable and calls handleConnection in its run() method to make a new thread each time a new client connects.
    - @throws IOException if the main server socket is broken.
- `public static void handleConnection(Socket socket) throws IOException;`
    - Handle a single client connection. Returns when client disconnects. Calls handleClientRequest()
    - @param socket socket where the client is connected
    - @throws IOException if connection has an error or terminates unexpectedly
- `public static String handleClientRequest(String input);`
    - handler for client input. Make requested mutations and return appropriate message to user.
    - @param input, the input from the client, from the grammar
    - @return the message (from the grammar) to the client
- `public static void main(String[] args)`
    - Start a server running
    - @param args contains the desired port for the server
- `public static void runServer(int port) throws IOException`
    - start a server running on specified port
    - @param port, the port to use
- `private String addUser(String user, Socket socket)`
    - adds a user to the infoMap unless it is a duplicate username, in which case it sends a INVALID_USER message
    - @param user, the user info to add (username and color separated by space).
    - @param socket, the socket this user is connected by
    - @return ONLINE message
- private String getOnlineUsers()
    - returns message containing all usernames and colors of online users separated by spaces (i.e. “Dan chartreuse Jenn brown Marianne blue”)
    - @return ONLINEUSERS message
- `private String logout(String username)`
    - deletes a user from the infoMap.
    - @param user, the username to be deleted
    - @return OFFLINE message
- `private String startConvo(String convo_id)`
    - @param convo_id, the convo_id for the conversation to start

    - @return START_CONVO message
- `private String updateConvo(String convo_id)`
    - @param convo_id, the convo_id of the conversation to change.
    - @return UPDATE message
    - called when the server receives a ADD_MSG message.
- `private String closeConvo(String convo_id)`
    - @param convo_id, the convo_id to delete from the universal list
    - @return CLOSE_CONVO message

UserInfo: Object to hold information about other users. First constructor is what the ChatServer class uses for its infoMap, and the second is for Users to keep track of other Users' names and colors.

```
public class UserInfo(String username, Color color, Socket socket);
public class UserInfo(String username, Color color);
```

- `private final String username;`
    - the username of the user
- `private final Color color;`
    - the color associated with the user
- `private final Socket socket;`
    - the socket used by the user
- `public String getUsername();`
    - @return the username
- `public Color getColor()`
    - @return the color
- `public Socket getSocket()`
    - @return the socket

UserGUI:

```
public class UserGUI extends JFrame;
```

- `public UserGUI(User user);`
    - `private final User user;`
        - `the User associated with the GUI`
- `public MainWindow()`
    - Creates a LoginView, displays to User
    - When confirmation is received from server, creates a ConversationView and displays to User
- Has ActionListeners for each field that call methods in User to relay the information

to the server.

- Ex:
    - Listener for the “Send” button: calls AddMsgToConvo, using user.activeConvo as the Conversation and the text in the text field as the text

LoginView:

`public class LoginView extends JPanel;`

- `private final JTextField ipAddress;`
- `private final JTextField portNumber;`
- `private final JTextField username;`
- `private final JColorChooser colorChooser;`
- `private final JDialog errorDialog;` //in case of invalid username

ConversationView:

`public class conversationView extends JPanel;`

- `private final JTabbedPane tabby;`
- `private final JScrollPane scrolly;`
- `private final JTable messages;`
- `private final JButton logout;`
- `private final JLabel newConvoLabel;` //”select a user to chat with”
- `private final JComboBox userlist;`
- `private final JButton newConvoButton;`
- Appropriate listeners for each of these JComponents

Client/Server Protocol

Client to server:

MESSAGE :== ADD_MSG | START_CONVO | CLOSE_CONVO | LOGIN | QUIT
LOGIN :== -l USER_INFO
QUIT :== -q USERNAME
ADD_MSG :== -c CONVO -u USERNAME -t TEXT //-u is the sender
START_CONVO :== -s CONVO_ID
CLOSE_CONVO :== -x CONVO_ID
CONVO_ID :== USERNAME+ //alphabetized
USER_INFO :== USERNAME COLOR
USERNAME :== [a-zA-Z]{1,10} //10 character limit, no numbers
COLOR :== [“red” “orange” “yellow” “green” “blue” “indigo” “violet” “chartreuse” “grey”]
TEXT :== [^NEWLINE]+

NEWLINE :== “\r?\n”

Server to client:

MESSAGE :== START_CONVO | UPDATE | CLOSE_CONVO | ONLINE | OFFLINE | INVALID_USER | ONLINE_USERS
ONLINE_USERS :== -f (USER_INFO)* //list of all online users’ names and colors
INVALID_USER :== -i USERNAME //error sent if username is already in use
ONLINE :== -o USER_INFO //lets clients know a new user is online
OFFLINE :== -q USERNAME //lets clients know a user has logged off
START_CONVO :== -s CONVO_ID //sent to recipient when new convo started
CLOSE_CONVO :== -x CONVO_ID //sent to other participant when one person exits
UPDATE :== -c CONVO_ID -u USERNAME -t TEXT //-u is the sender
CONVO_ID :== USERNAME+   //alphabetized
USER_INFO :== USERNAME COLOR
USERNAME :== [a-zA-Z]{1,10} //10 character limit, no numbers
COLOR :== [“red” “orange” “yellow” “green” “blue” “indigo” “violet” “chartreuse” “grey”]
TEXT :== [^NEWLINE]+
NEWLINE :== “\r?\n”