Assignment 2 Design Document
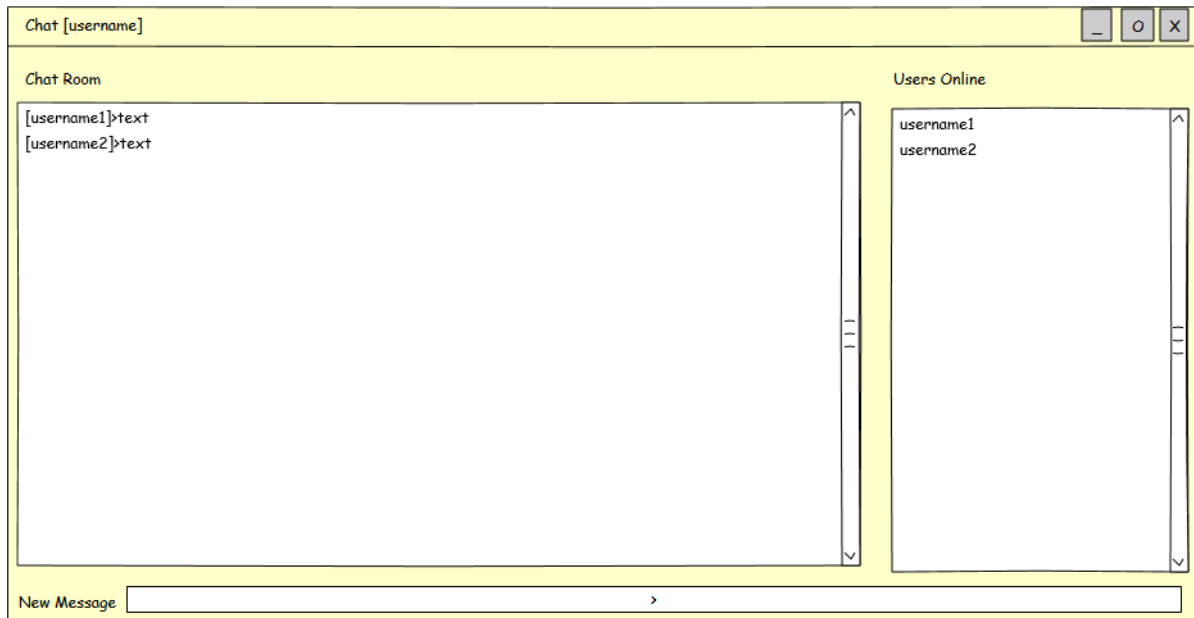
Chat Client GUI View Design Proposal

Figure 1 GUI Design

Chat GUI View should have 3 main parts:

1. **Char Room text pane**: where user will see all the messages from all connected users appearing from bottom up. Latest message will appear on top of the pane.
2. **Users Online text pane**: where user will see all the connected users
3. **New message text field**: where user can write a new message and on ENTER message will be send to server and populate on every connected user Chat Room text pane

Chat Room message proposed format

After some research done it seems that chat rooms have their standard format.

[icon][*username*]>message text

Where icon and username is chosen by user at client application startup. Username is given random color at startup as well. For this to function I will need to pass a state to server which contain the icon, username, color and text message.

For this I will create a simple java bean which will represent this message format and be passed between the clients and the server.

```java
class Line implements Serializable{

    private final Icon icon;
    private final Color color;
    private final String name;
    private final String text;

    public Line(Icon icon, Color color, String name, String text) {
        this.icon = icon;
        this.color = color;
        this.name = name;
        this.text = text;
    }

    Icon getIcon() { return icon; }
    Color getColor() { return color; }
    String getName() { return name; }
    String getText() { return text; }
}
```

Proposed interface of GUI view

```java
interface ChatView {
    void postMessage(Line line);
    void updateConnectedUsers(List<Client> onlineClients);
}
```

Figure 2 GUI interface

Where **postMessage()** method will post proposed message in Chat Room text pane and **updateConnectedUsers()** method will fill the Online Users text pane with clients names from **onlineClients** list. I will build this GUI using **javax.swing** library.

Chat Server Design proposal

For a client to be able call a server's methods remotely a server must implement an interface with every method throwing RemoteException. The server class must also inherit from UnicastRemoteObject class.

Server will be just passing message lines to all the client.

```java
interface Server extends Remote {
    void connect(Client client) throws RemoteException;
    void disconnect(Client client) throws RemoteException;
    void send(Line line) throws RemoteException;
}
```

Figure 3 Chat Server interface

Where **connect()** method will connect the client to server. Method **disconnect()** will disconnect the client from server and **send()** will send the client state to server.

Chat Client Design proposal

For the server to call a client's methods remotely a client must implement an interface with every method throwing RemoteException. The client class must also inherit from UnicastRemoteObject.

Client will be just a proxy for GUI interface plus will provide the client state getters for server to have access to if needed.

```java
interface Client extends Remote {
    void postMessage(Line line) throws RemoteException;
    void updateConnectedClientList(List<Client> connectedClients) throws
RemoteException;
    String getUserName() throws RemoteException;
    Color getColor() throws RemoteException;
    String getText() throws RemoteException;
    Icon getIcon() throws RemoteException;
}
```

Figure 4 Chat Client interface

Where postMessage() will send a message for client to insert into Chat Room text pane. Method updateConnectedClientList() will send a list of connected clients for a client to update its Online User text pane. Rest of the methods are getters for the current client state. Color, username and icon are constants created at a client startup and text is actual message which can change.

How this will work?

It will be a simple observer pattern implementation using Remote Method Invocations.

A client will remotely invoke **server.connect(client)** method which will add the client into a connected clients collection on the server. Then on **server.send(line)** method invocation the server will pass the client object state to all clients from the collection. On **server.disconnect(client)** method invocation the server will remove the given client from the collection. The **connect()** and **disconnect()** methods should be **synchronized** because they modify the server's list of connected clients.
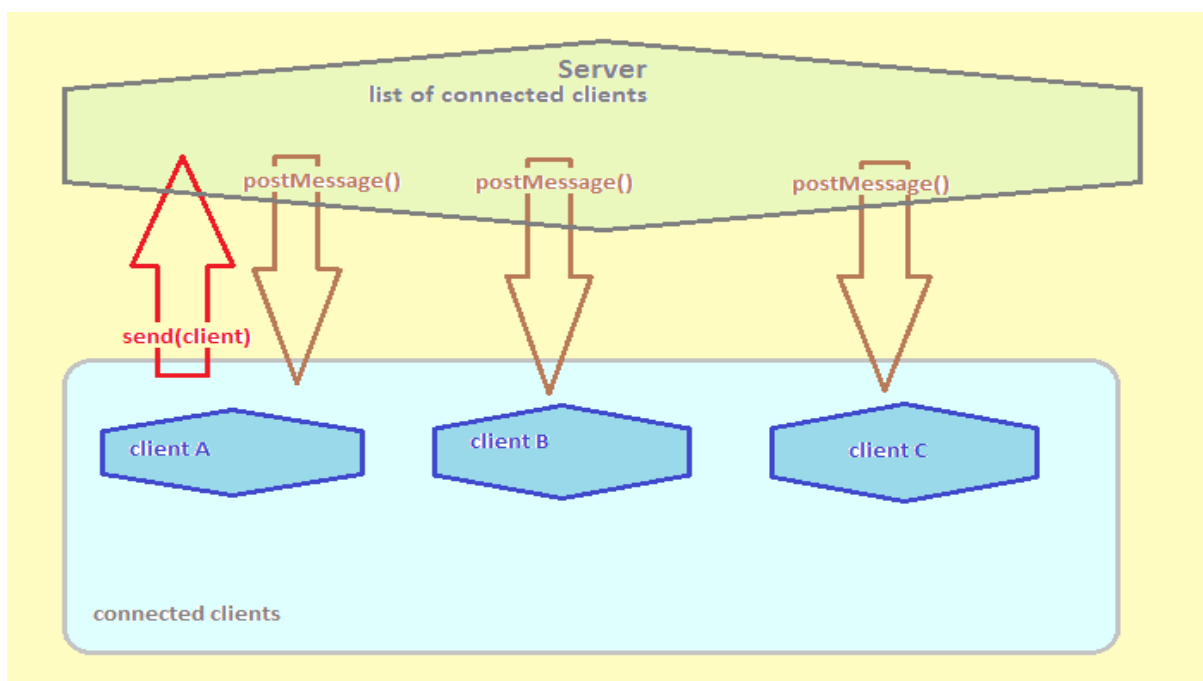

Figure 5 Observer pattern visualization

Conclusion

The proposed design was proven quite solid. Application was developed smoothly without any serious problems and without a significant deviation from the design. The biggest challenge was to implement the GUI view.

I added a functionality of the server remembering last 20 lines of the chat messages, so when new client is connected he sees last 20 lines of ongoing conversation.
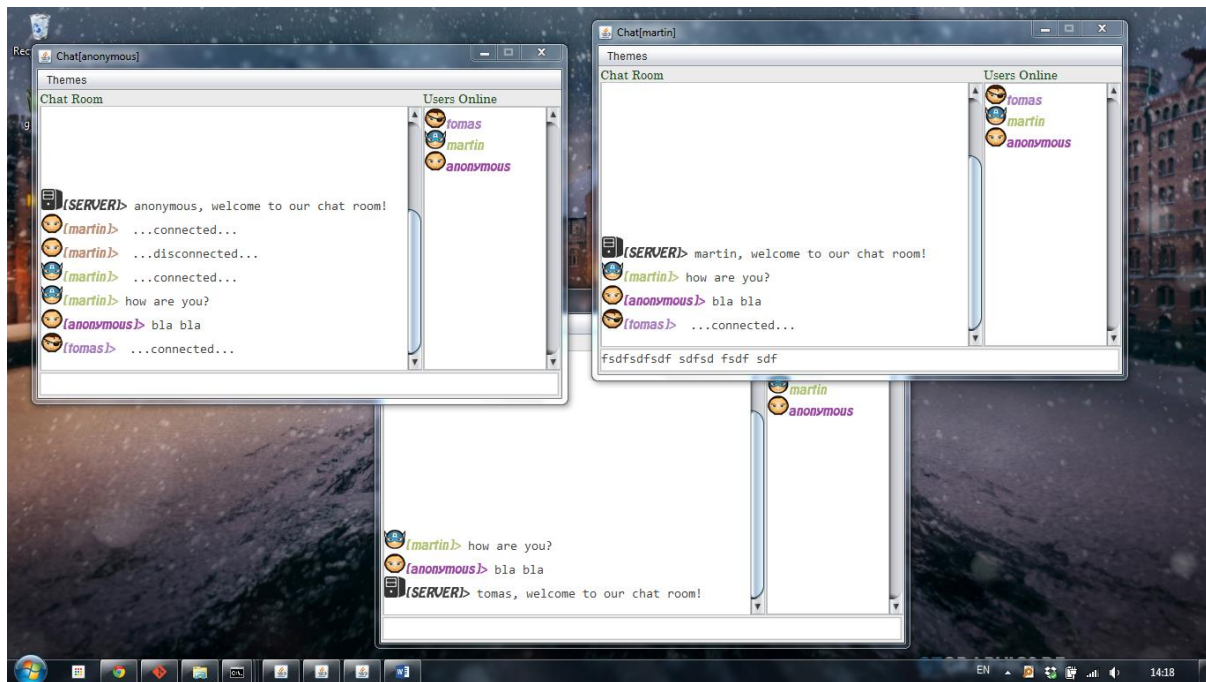

Figure 6 Chat Application implementation