Project Specification – MultiText

Names: Ethan Chen, Vinay Venkatesh

Period 6

Overall Description

MultiText is a multi-user chat room.

In this project we will program our own multi-user online chat room.

The chat room keeps a list of the history of messages sent as well as a list of who is online.

When a message is sent, the recipient can see the messages coming in in real time. Each message contains the text contained in the message sent, the username of the user who sent it, its index in the chat, which is not displayed, and at what time the message was sent.

Users can also identify what other workers are online through a small menu on the side, listing their usernames. The server updates the list whenever a user logs on and off, adding or removing their username from the list.

Class/Interface Overview

Top Level Class

Multitext – the main class of the application. Creates the GUI and runs the application. Creates the Server which allows Workers to log in. Also creates the Chat which stores all messages and online users.

Major Classes

Server – Where the client connection is made and accepted and online users and chat are contained

ServerMain – Where the server is created given a port and the server is started

ServerWorker – Where connections are handled and the login mechanism is, and where Worker statuses are set, as well as the functionalities of each Worker

ChatClient - Where the client connects to the server and users can send and receive data to and from the server

GUI Classes

UserListPane – GUI for the Worker where other online users are visible

MessagePane – GUI for two workers to directly message each other

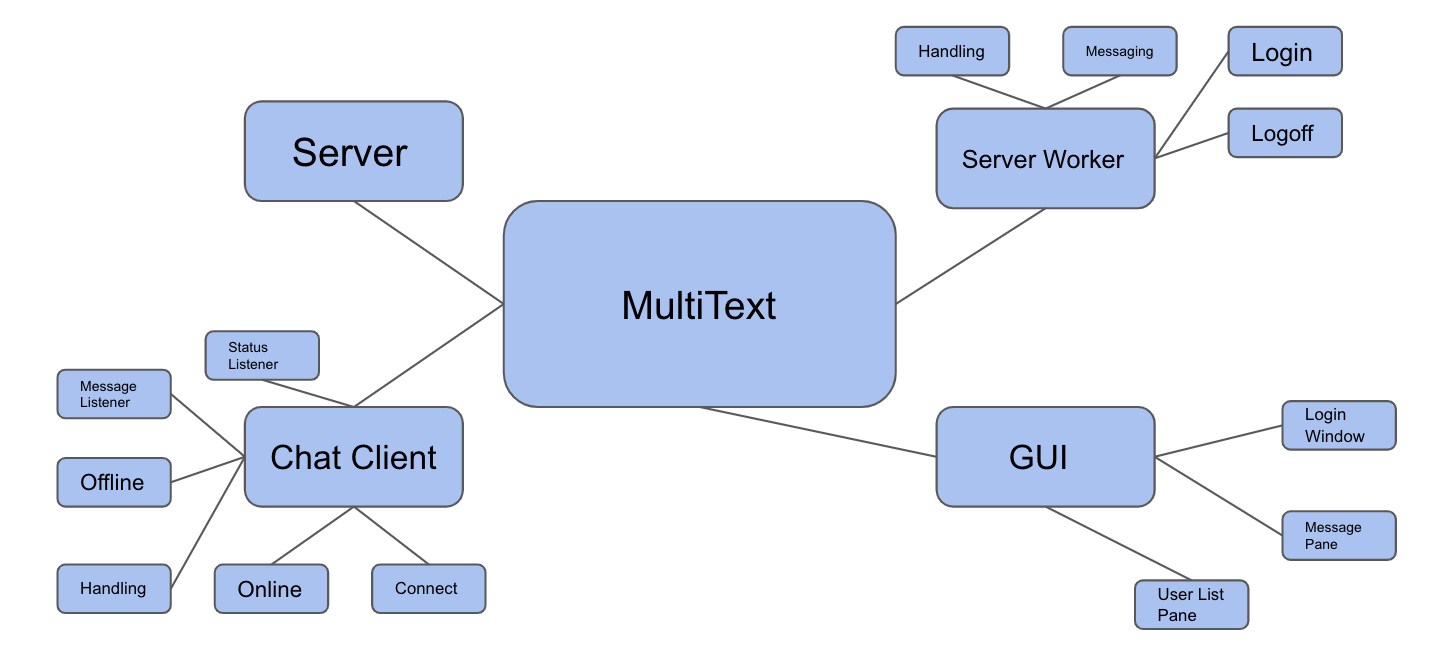
LoginWindow – GUI for the logging in of a Worker. Accepts a username and password.

Interfaces

MessageListener - Interface which executes a response when a message is received

UserStatusListener - Interface which executes a response when a user is online or offline

Rough Class Diagram



Structural Design

The following data structures will be used.

| **Description** | **Structure** |
| --- | --- |
| Logged-In Workers | List<ServerWorker> |
| Chat Message List | ArrayList<Message> |
| User Status Listeners | ArrayList<UserStatusListeners> |
| Message Listeners | ArrayList<MessageListener> |

Data structure rationale

We chose an ArrayList to contain the logged in workers because it was simple to display the Workers in the GUI and also could add and remove different Workers very easily, and since efficiency is a very small issue when dealing with small scale things, adding and removing to the ArrayList is not an issue.

We also chose an ArrayList to contain the message list again because of its simplicity and ease of use.

High Level Major Class Specifications

Server

- Attributes

o int serverPort

o ArrayList<ServerWorker> workerList

- Methods

o void run()

o List<ServerWorker> getWorkerList()

o void removeWorker()

ServerWorker

- Attributes

o Socket clientSocket

o String login

* Server server
* OutputStream outputStream

- Methods

o void run()

o void handleClientSocket()

o void handleMessage()

* void handleLogoff()
* String getLogin()
* void handleLogin()
* void send()

ServerMain

- Methods

o void main()

LoginWindow

* Methods
* void doLogin()

UserListPane

* Methods
  + void online()
  + void offline()

MessagePane

* Methods
* void onMessage()

ChatClient

- Attributes

o String serverName

o String serverPort

* Socket socket
* OutputStream serverOut
* InputStream serverIn
* BufferedReader bufferIn
* private ArrayList<UserStatusListener> userStatusListeners = new ArrayList<>();
* private ArrayList<MessageListener> messageListeners = new ArrayList<>();

- Methods

o void main()

o void msg()

o void logoff()

* void login()
* void handleMessage()
* void startMessageReader()
* void readMessageLoop()
* void handleOffline()
* void handleOnline()
* boolean connect()
* void addUserStatusListener()
* void removeUserStatusListener()
* void addMessageListener()
* void removeMessageListener()