Design and implementation of a Whiteboard that can be shared between multiple users over the network

DISTRIBUTED SYSTEMS

Distributed Shared Whiteboard

Darth Vaders
Abhinav K Singh(1037729)
Gagan K Prasanna(1101741)
Mengyi Fan

COMP90015: DISTRIBUTED SYSTEMS

Semester 2,2019

Assignment 2

Problem Statement

Design a Whiteboard that can be shared between multiple users over the network. Multiple users are allowed to draw on a shared interactive canvas.

Overview

Designing and implementing a shared whiteboard involves separating the functionality into two major parts: Whiteboard and the server. The user starting the whiteboard becomes the whiteboard manager. A notification is sent to the manager when other users want to join. The manager has the ability to kick someone out anytime. The whiteboard interacts with the server on user's behalf by sending any images drawn on the whiteboard to be broadcasted on the screen of other users.

Component Description

Client (Whiteboard)

Client program has two important tasks -

1. Interacting with the user through a Graphical User Interface (GUI)
2. Interacting with the server on behalf of the user by using either TCP or UDP mechanism

Client program requires an interface for interacting with the end users. User interaction involves capturing user's input on the whiteboard and then transmitting it to the server.

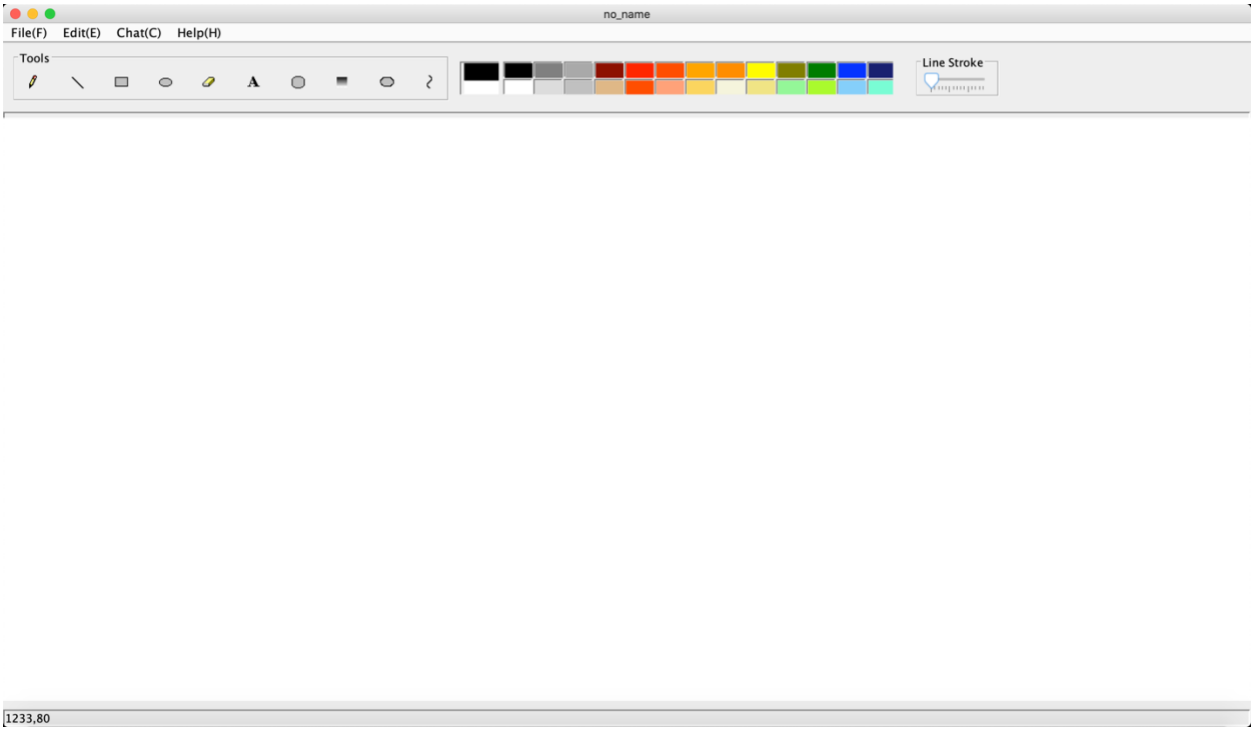


Figure 1: insert whiteboard screenshot

GUI allows the user to interact with the whiteboard with the following tools like **Pencil, Color, Shapes, Eraser and Text Insertion.** The tools selected by the user and the position on the whiteboard where the modifications are made are captured using the mouse tracking function. The user also has the options to open a **new file, save** the current file and use **the chat feature**. The exit button is used to disconnect the user from the server.

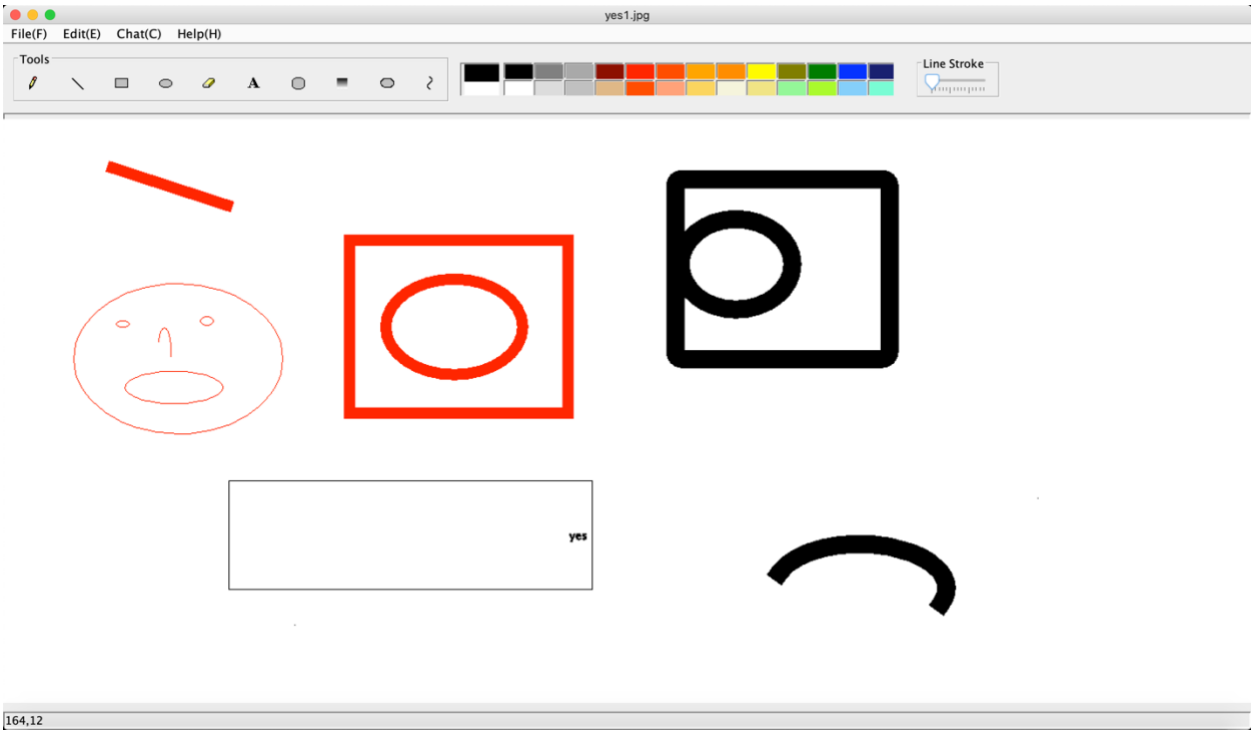


Figure 2: GUI in action

Client manages interaction with the server on behalf of user.

Server

Server program has the following tasks-

1. Managing multiple client threads in a way that concurrency is achieved
2. Storing and extracting data from the data structure
3. Interacting with the whiteboard manager using an interface

The whiteboard sever uses **thread-per-connection** mechanism for supporting multiple clients and the reason for choosing this particular mechanism is explained in multithreading section below.
Dictionary server uses **List data format** for storing the modifications done to the whiteboard. This data is used to recreate the image on the whiteboard when new users join.
The GUI of the server interacting with the whiteboard manager is shown below

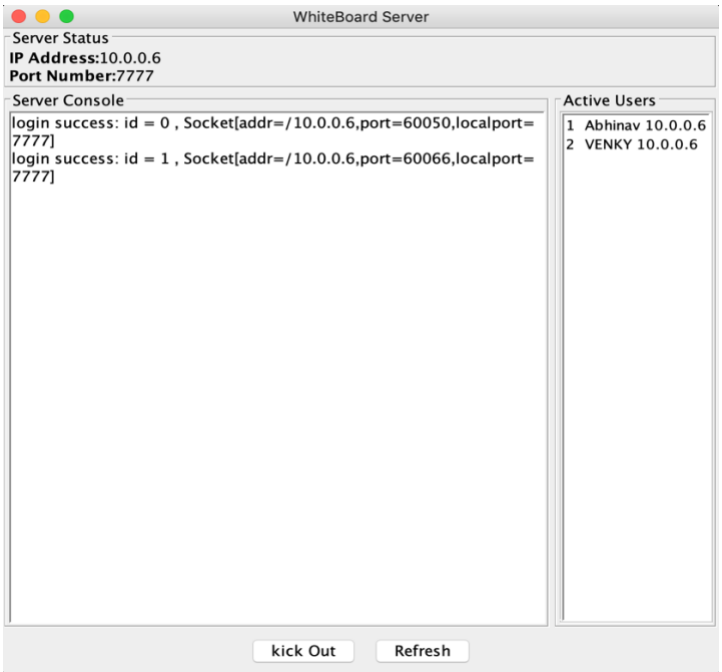


Figure 3: Server GUI

GUI captures port number, the IP address and shows the list of active users. The whiteboard manager can use the **kick** to remove any user. The **refresh** button can be used to refresh the list of active users when the kick action has been carried out.

Multithreading

Server manages multiple client requests by using multithreaded server architecture. The advantages of using multi-threaded server compared to a single-threaded server are mentioned below-

1. Server program spends minimum time outside the accept() call
2. Long client requests will not block the entire server
3. Concurrency is guaranteed

There are multiple mechanisms for supporting multithreaded server. Whiteboard server uses thread per connection model and the reasons for using the same is mentioned below-

- It amortizes the cost of spawning the thread across multiple request generated from the same client process
- It is simple to implement and performs efficiently for small number of users performing multiple requests. As the number of users using the dictionary servers are small thread per connections suits the premise
- Main disadvantage of the thread per connection architecture is that threads remain active during inactivity on client's end, but this condition is mitigated as the users can disconnect by using disconnect button if they no longer require services provided by the server thereby killing the thread spawned for that particular user(connection) on the server side

Thread per connection architecture is only suited for the current premise where the number of users is limited as this architecture doesn't scale well. Its limitation is its inability to balance load under extreme conditions creating performance bottle necks.

Data Structures

Data structures used by the application are mentioned below-

- List- The list is used to save the modifications sent by the users. When a new user joins the whiteboard the data from this list is used to recreate the existing image on the whiteboard

CLASS DESIGN

The entire class design can be separated into three parts –

1. Whiteboard (Client) design-

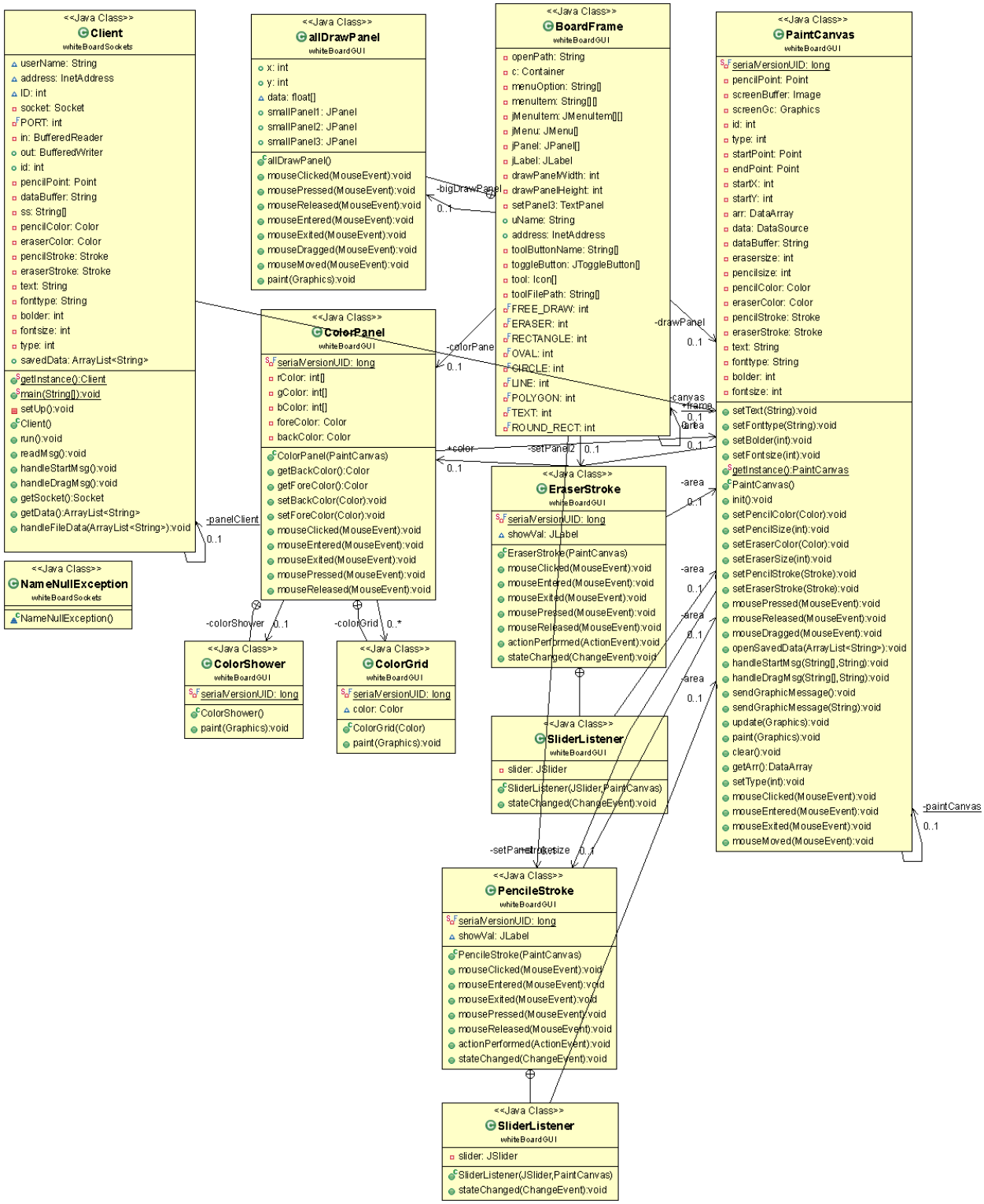


Figure 4: Client's UML Diagram

2. Server class design –

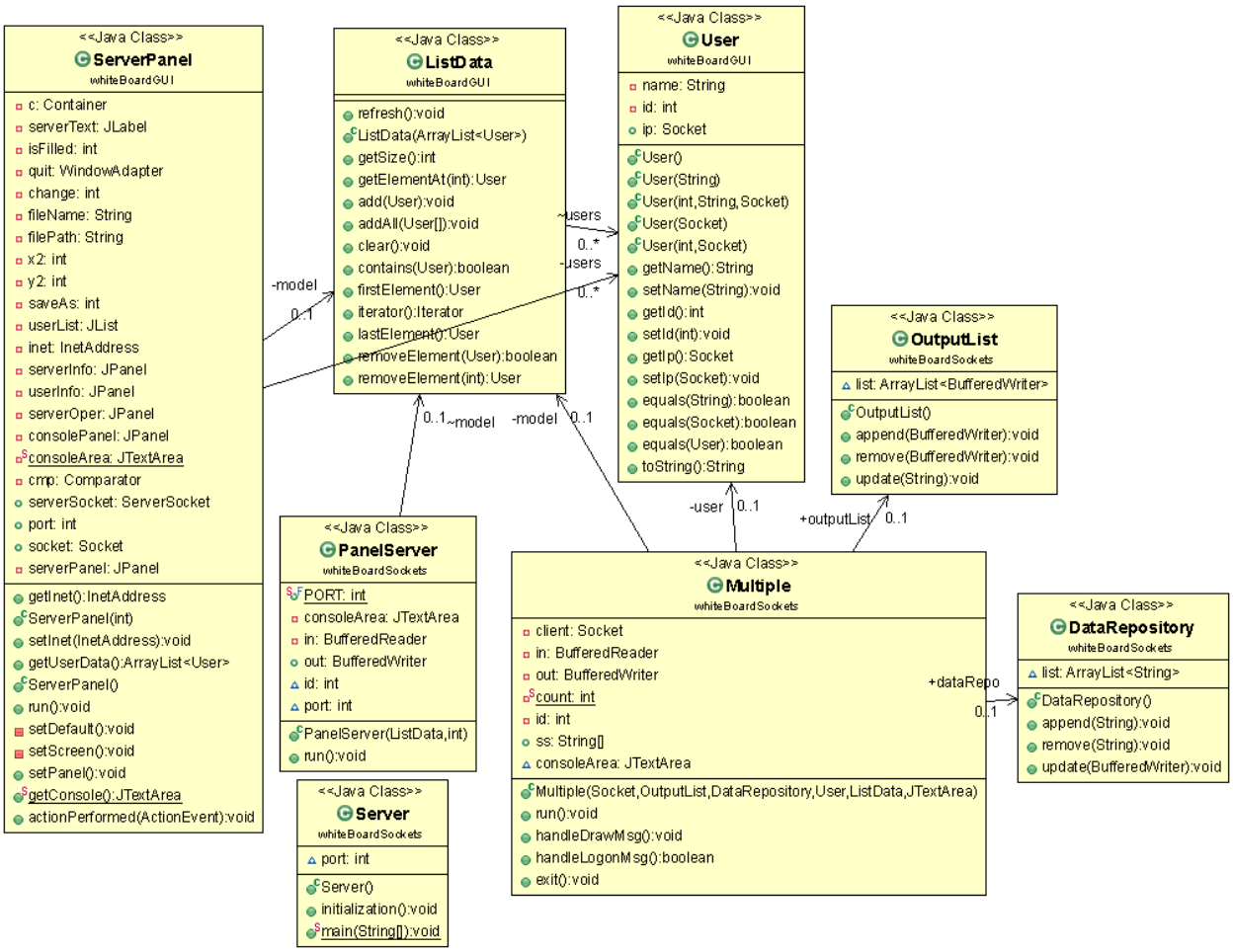


Figure 5: Server's UML Diagram

3. Chat panel design-

The chat panel allows users to talk with each other using text messages. The client UI consists with 2 JTextAreas, one for displaying the chat messages, one for editing the message you want to send.

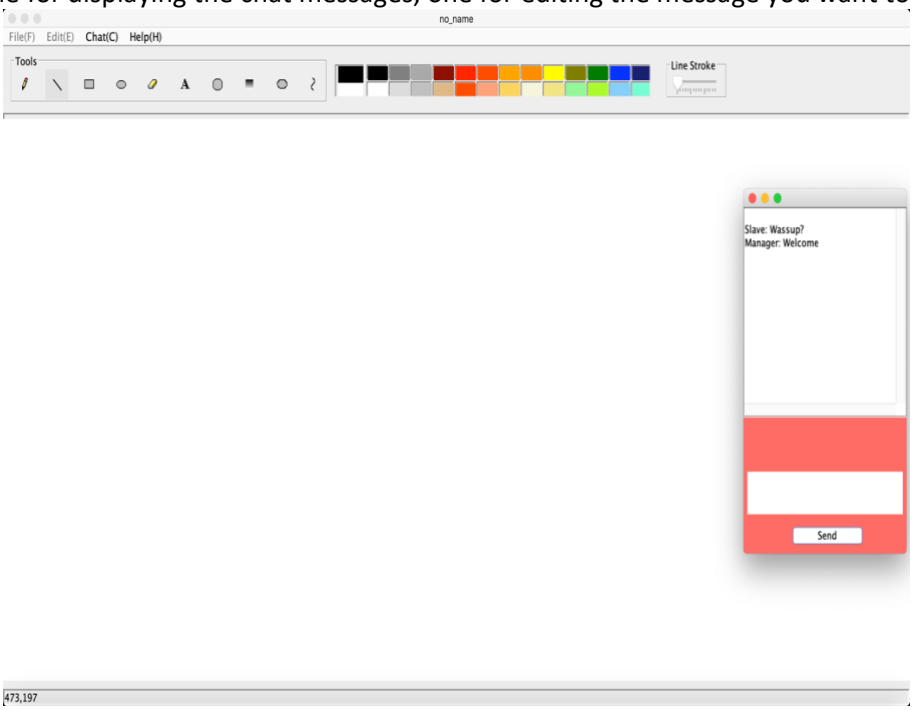
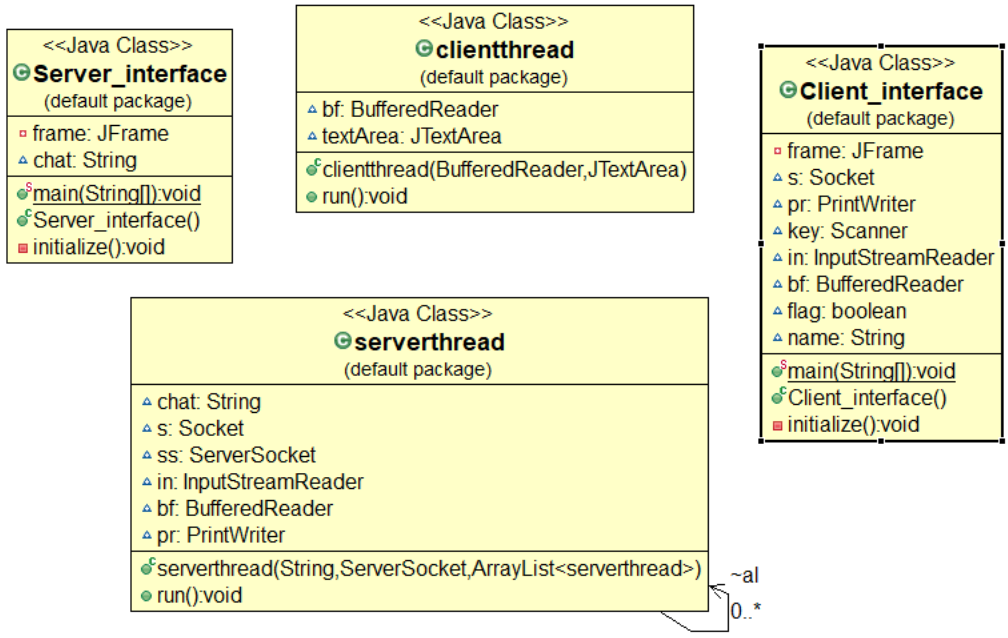


Figure 6: Slave Client with Chat Window

The chat panel uses socket and threads. It uses thread per connection. It has a server part and a client part. Once the server starts, each client can press the connect button to connect with the server. Once a client sends a message to the server, the server will send the same message to all clients that is connect with the server.
Here is the class design for the chat panel:



Configurations

The client application has two important configurations which are mentioned below:
Manager: The manager client configuration enjoys greater control over the whiteboard operations when compared to the slave client configuration. Manager is capable of performing features like New, Open, Save and SaveAs. Manager configuration can kick clients with slave configuration by using server's GUI. ServerGUI and Manager client configuration work in tandem and complement each other for controlling the whiteboard.
Slave: The slave client configuration is almost similar to the manager configuration but with lesser privileges and control over the whiteboard operations. Slave can't perform special operations like New, Open, Save and SaveAs.

INTERACTION

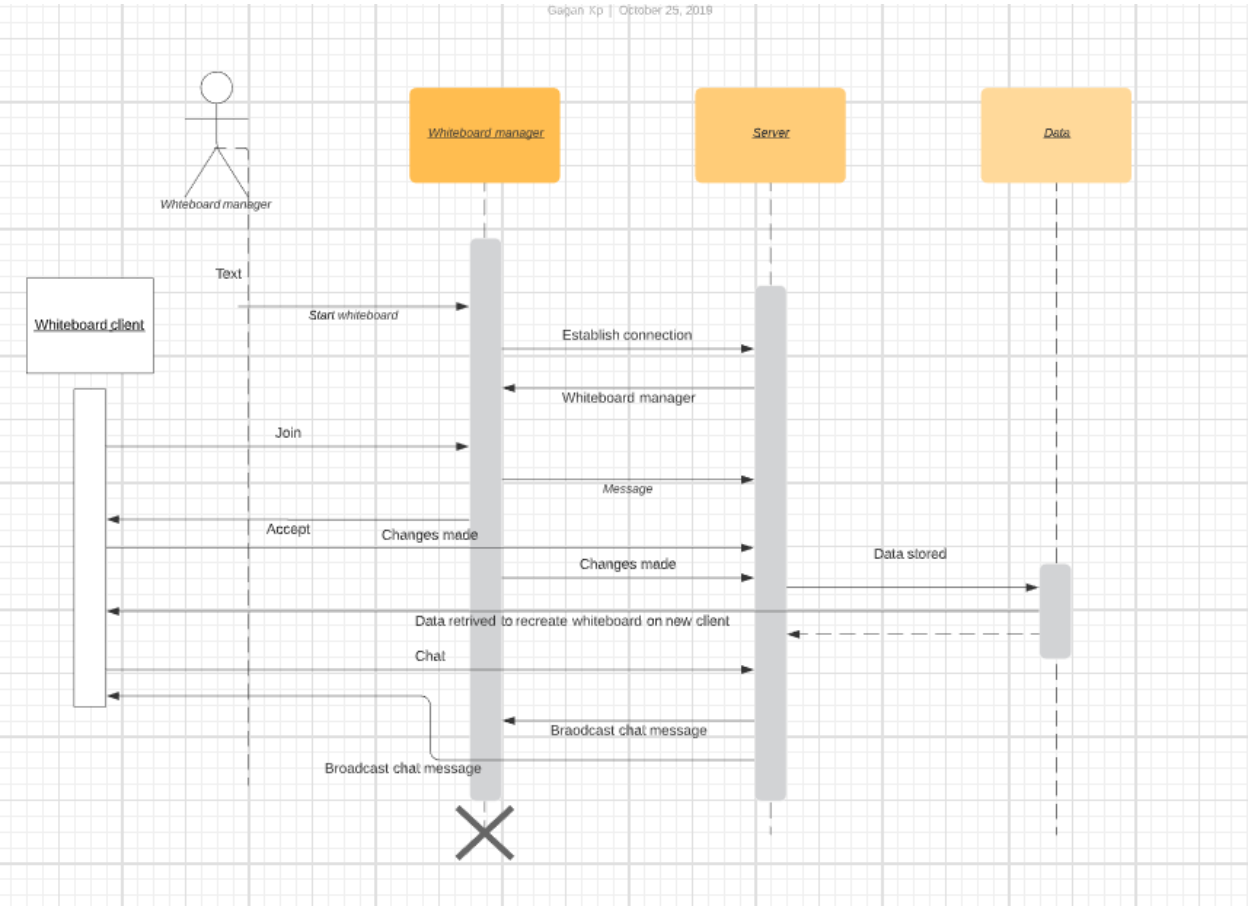
TCP

TCP mechanism is preferred over UDP because of the following reasons –

1. TCP ensures reliability while sending messages whereas there is no reliability guaranteed while using UDP
2. TCP ensures ordering of messages as they are delivered therefore eliminating chances of receiving an out of order packets
3. TCP is connection oriented and suited for point to point data transmission where as UDP is connectionless and is generally used for broadcasting purpose.

Overall Flow

The first user to start the whiteboard becomes the whiteboard manager. Connection to the server is established. The modifications made on the whiteboard are sent to the server. It gets broadcasted to all the users connected and is also stored in the data repository. A new user who wants to join the whiteboard session sends a request to the whiteboard manager. The manager can allow him to join or deny the request. Once the user is connected the data from the server is sent to the new user to recreate the images on the current whiteboard.



Creativity

Precise Concurrency

The clients use precisely built multi-threaded system allowing it to display objects to all the users with negligible delay thereby enhancing the real time user experience. Multiple client can perform the operation of drawing different shapes at the same time without any visible lag on the whiteboard screen.

Smart Chat

The chat has been designed in such a way that it can be used to perform some operations as well. For example the chat can be used to retrieve the name of all active members using the whiteboard by using the command “activeusers”.

Figure 7: Chat Window Showing Active Users

Swing workers

Swing workers are used while designing both server and client GUI. Swing workers prevent other program threads from interrupting or freezing the GUI thread by creating separate threads for any non-GUI operation performed in the GUI thread. Separate threads ensure that GUI will never get stuck even in case of errors in non-GUI operations.

```
new SwingWorker()
{
    protected Object doInBackground()throws Exception
    {
        server.finalize();
        return null;
    }
}.execute();
```

Figure 8: Swing Worker Stub

Contributions

Name	Contribution Area	% Contribution
Abhinav K Singh	Multithreaded Structure, Backend dataflow, Code Integration, Report	50%
Gagan Kuthi	Error Handling, Data Structures, Shapes, Report	40%
Mengyi Fan	Multiple Client Chat	10%

Inspiration

Creating shared whiteboard application was an enriching experience and it would not have been possible without the help from external sources. External Sources such as Git, GitLab, stack overflow and Wikipedia helped in getting ideas for better and effective implementation of whiteboard features. During the deadlocks answer from developers across the world helped us in solving the problem in an effective manner. All ideas used or implemented are from open source platforms and hence not subject to copyright.

Conclusion

Distributed shared whiteboard was established with features and components mentioned above. The final application satisfies all the requirements mentioned in the problem statement and performs as expected.