DISTRIBUTED WHITEBOARD REPORT

COMP90015 DISTRIBUTED SYSTEM

ZHUOLUN WU 954465

WHITEBOARD SYSTEM

The deliverable of this project is a shared whiteboard which allows multiple users to draw simultaneously on a canvas, and chat in real-time. The system uses TCP sockets for connection and being developed in two parts: Server and Client.

After the server gets started, the first client connected would become the manager of the system, who can use various features to manage not only the white board, also other connected users. The manager is the only one who can create, save, open from, and close the white board, and any new user's connection should be approved by the manager. The manager is also able to kick any connected user.

Command line argument for starting the server: java -jar Server.jar <port number>

Command line argument for starting the client: java -jar Client.jar

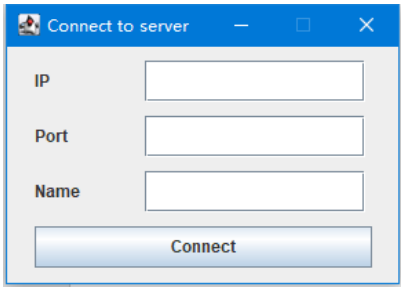


Figure 1. Client GUI for connection

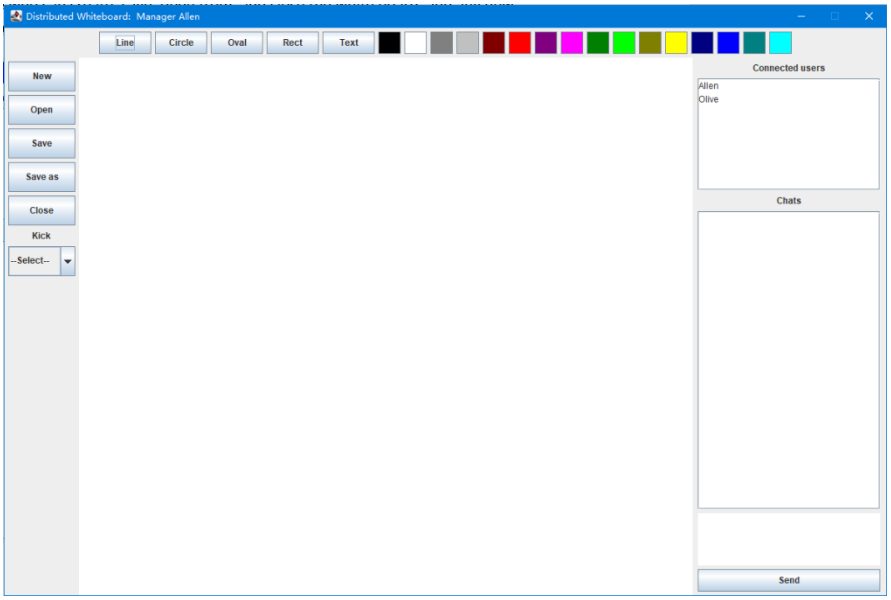


Figure 2. Manager's GUI

ARCHITECTURE DESIGN

This section would introduce architecture designs on both server side and client side, and their interaction sequences. Connection between both sides would be discussed as well.

SERVER

The server side of the whiteboard consists of one **Server** class and one Runnable **ServerThread** class (figure 3). When it starts running, Server will read the opening port from command line, and wait for connection, and a ServerThread object is created for each incoming connection request.

The ServerThread object is responsible for read and write messages into the socket for communicating with client. All incoming messages are processed within the ServerThread object. Once it needs to manipulate data (e.g., the ArrayList containing all other ServerThreads) in the Server object, it would always use static synchronized methods provided by Server class to solve the concurrency issue.

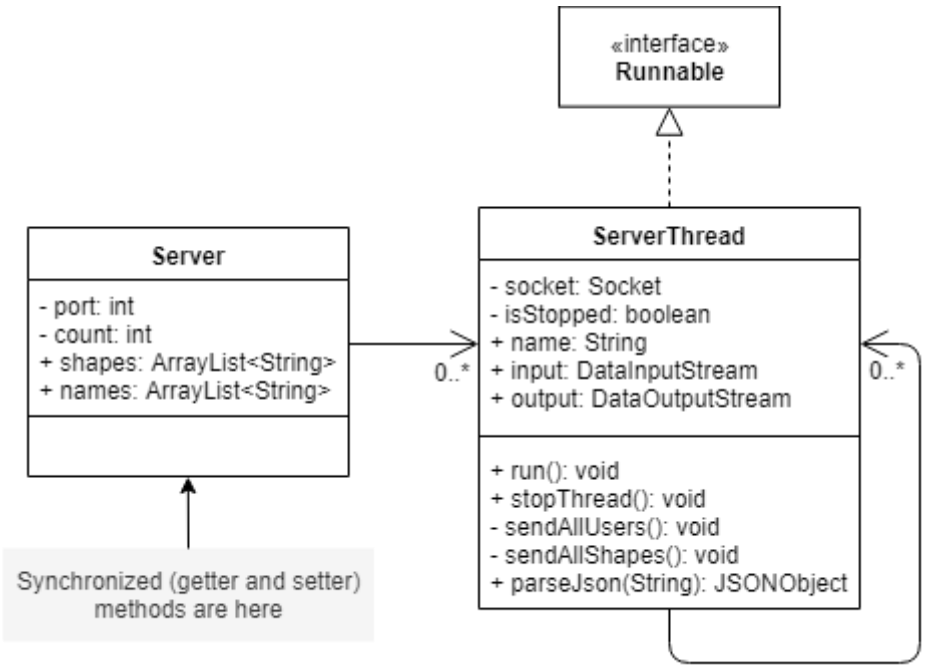


Figure 3. Class diagram for Server

When the server starts, the first client would be considered as the manager, and the rest need manager's approvement to connect to the whiteboard (figure 4). Also notice processes ① and ② in the diagram, if the manager rejects, process ① would become stopThread(), and process ② would be skipped.

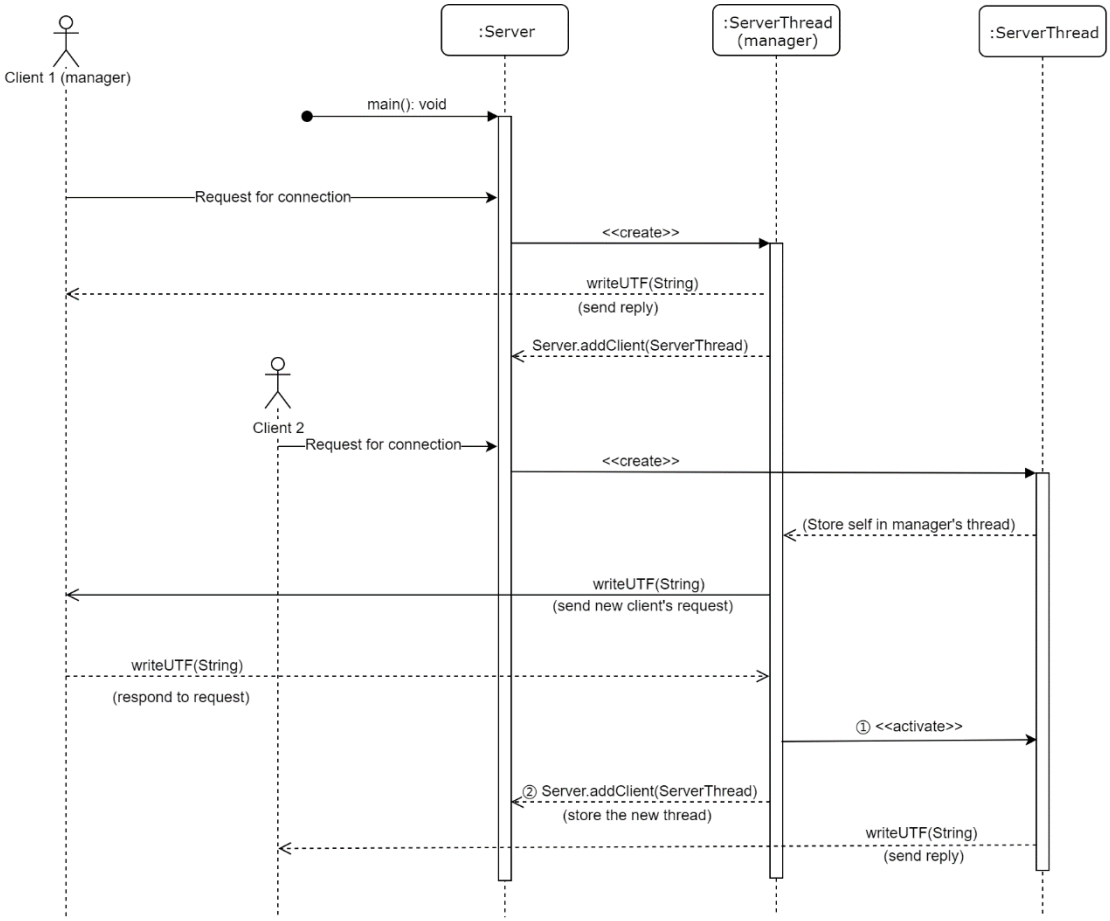


Figure 4. Sequence diagram for Server

In short, after the server started running, Server would create a ServerThread for client 1 (manager), inform client 1 is the manager by sending a reply in the thread, and store the thread in Server (during runtime).

If the second client tries to connect, another thread is created for him. Then this thread would use manager's thread to send the request to the manager, and store self into manager's thread. Once the manager thread received reply from the manager, it will either activate client 2's thread and add it to Server's storage or stop the thread.

As for how to process messages, would be discussed in the Communication section.

CLIENT

For client side, there is one **Board** which implements GUI designs, a **Listener** added to each graphic element (JButton, Graphics2D, etc.) of the Board. There is a **ClientThread** for communicating with the Server, and a Shape class for storing necessary data for a shape drawn on whiteboard (figure 5).

In addition, unlike the server side, each client will only have one ClientThread created, where the ClientThread class is using the singleton pattern.

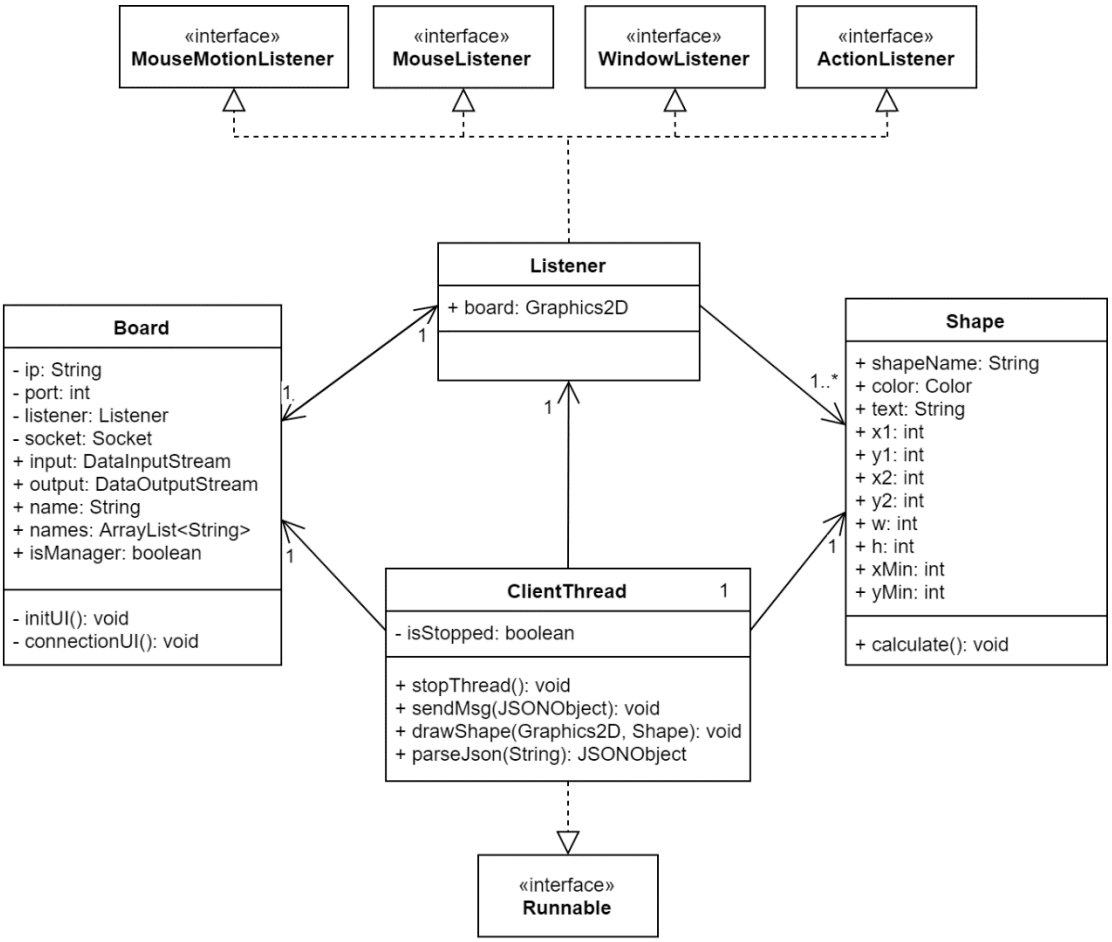


Figure 5. Class diagram for Client

Once the client tries to connect with the server, the request is made in the Board class. If this client is the first one connects the server, it would start rendering the whiteboard directly, otherwise it would need to wait for manager's approvement. Once the connection is made, all the rest work of communication would be left to the ClientThread.

As shown in figure 6, the client side starts by using IP address and port number collected (figure 1) to connect with the server. The server would send back if the user is manager or not, and Board needs manager's approve message to enter the whiteboard if the user is not manager.

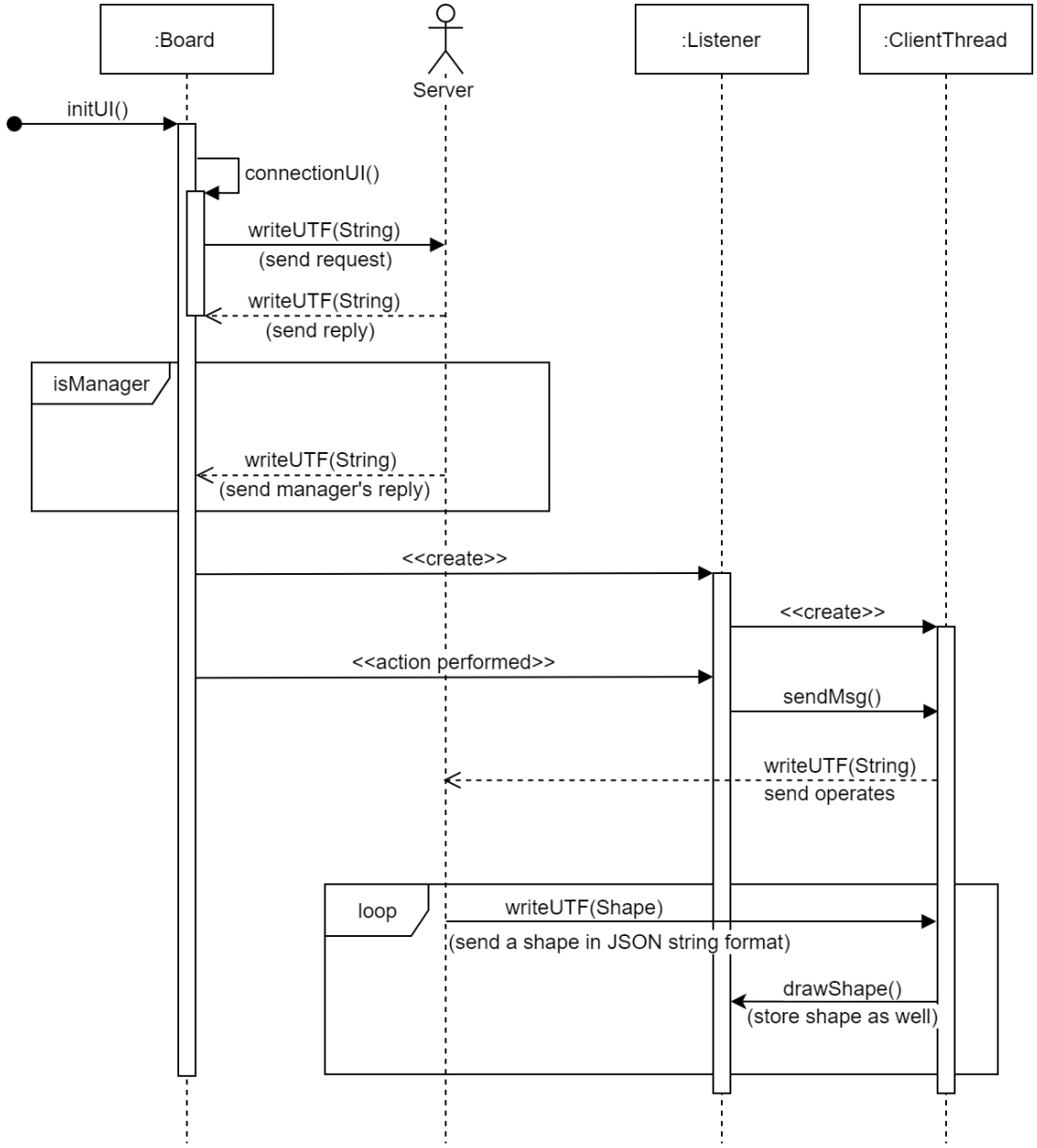


Figure 6. Sequence diagram for Client

The sequence diagram also shows two basic interaction logics. The first one is about user's operation on whiteboard. Any action would let the Listener use thread to send the action to server. The second is about client constantly receiving message of a shape and draw it on whiteboard. These two are abstracted presentation of how client works, any further requests and messages basically all work the same.

One thing needs to mention is that once the user draws a shape on whiteboard, the board would not draw it until it receives Server's broadcast of the shape that ClientThread send.

COMMUNICATION

There are two kinds of message being received and processed by the server, and there are two examples listed in the figure 6. Some messages are broadcasted, like someone draws a shape and everyone gets updated. Some messages are unicasted, like manager kicks a client, and the client gets kicked will receive the notice.

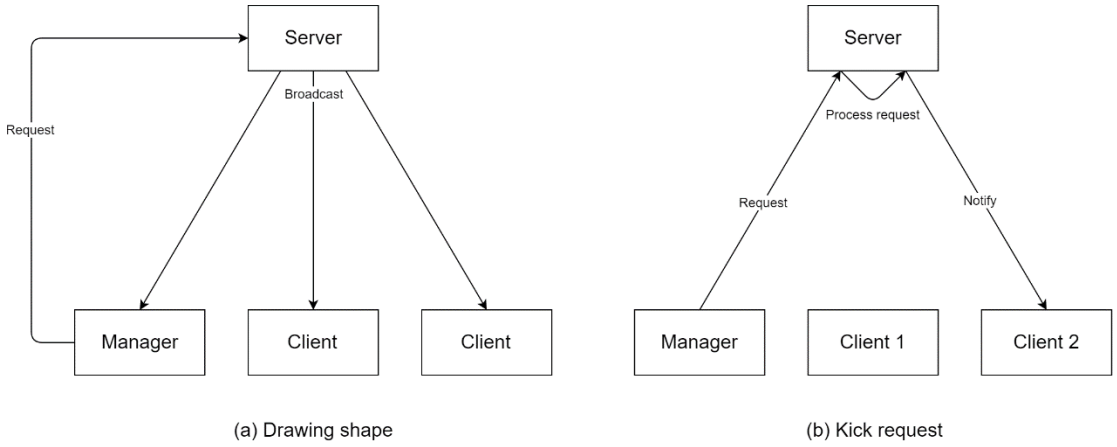


Figure 6. Simple communication examples

COMMUNICATION PROTOCOL

TCP protocol is used in the white board system, in purpose to support reliable and easy-to-use connection between server and client. There are many other protocols applicable to this system, and I will explain why TCP is selected.

Remote Method Invocation (RMI) protocol is built on top of sockets, but more concise and easier to develop. Message handling logic in a TCP or UDP communication system seems to be redundant when it comes to RMI, where each situation is solved in the form of remote method, and this becomes more readable and easier for maintenance.

However, RMI for this project has two side effects. The first is that RMI only for Java. Standard TCP/IP protocols allow server and client being developed in different language, or clients being developed using different language on different platform. RMI is restricted for Java platform only. The second is that using RMI is not good for future development. Communicate under RMI protocol does have issue about the overhead of marshalling and unmarshalling. Currently only single shapes can be drawn, but in the future if “pencil” or “free line” is being implemented, huge amount of data would get transmitted. Using RMI would cause some network occupancy and latency issues.

Comparing to UDP, TCP is more reliable, and can dynamically examine if client is still online – using try and catch on sending message is more efficient than sending a datagram back.

MESSAGING FORMAT

Both the server and the client send and receive JSON formatted string for communication. Each JSON object starts with a header to tell what request this is (e.g. {“header”:“new”} for creating a new canvas) and followed by other data (figure 7).

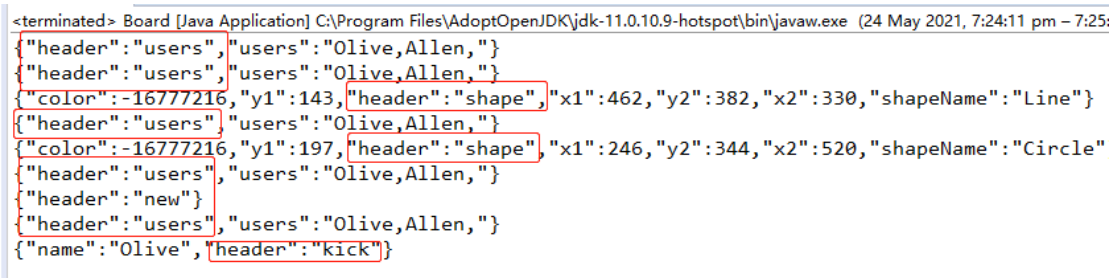


Figure 7. JSON formatted string message

PROCESSABLE MESSAGE

Server:

- Connect
- Connect reply (manager’s approvement)
- Initialize (send all drawn shapes to a newly connected client)
- Shape (any newly drawn shape, or the manager opens a whiteboard from a previous saved one)
- Chat
- New
- Close
- Kick

Client:

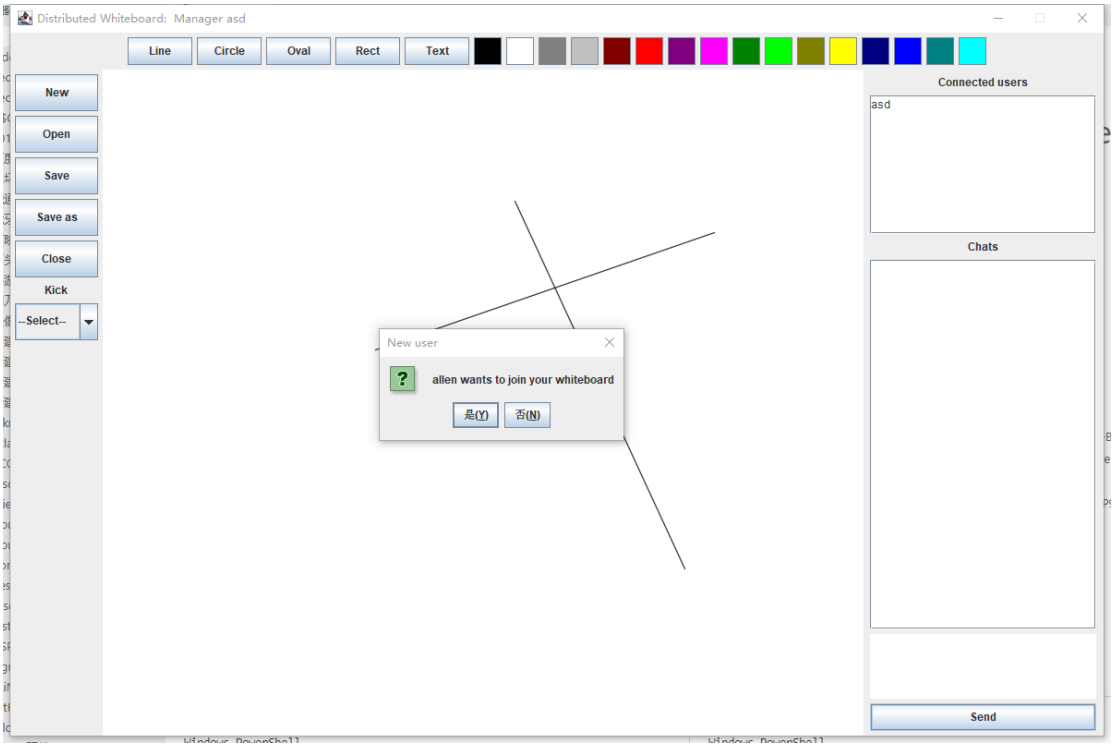
- Shape (Server’s broadcast)
- Connect (manager sends approvement message)
- Users (collection of all connected users)
- Chat
- New
- Close
- Kick

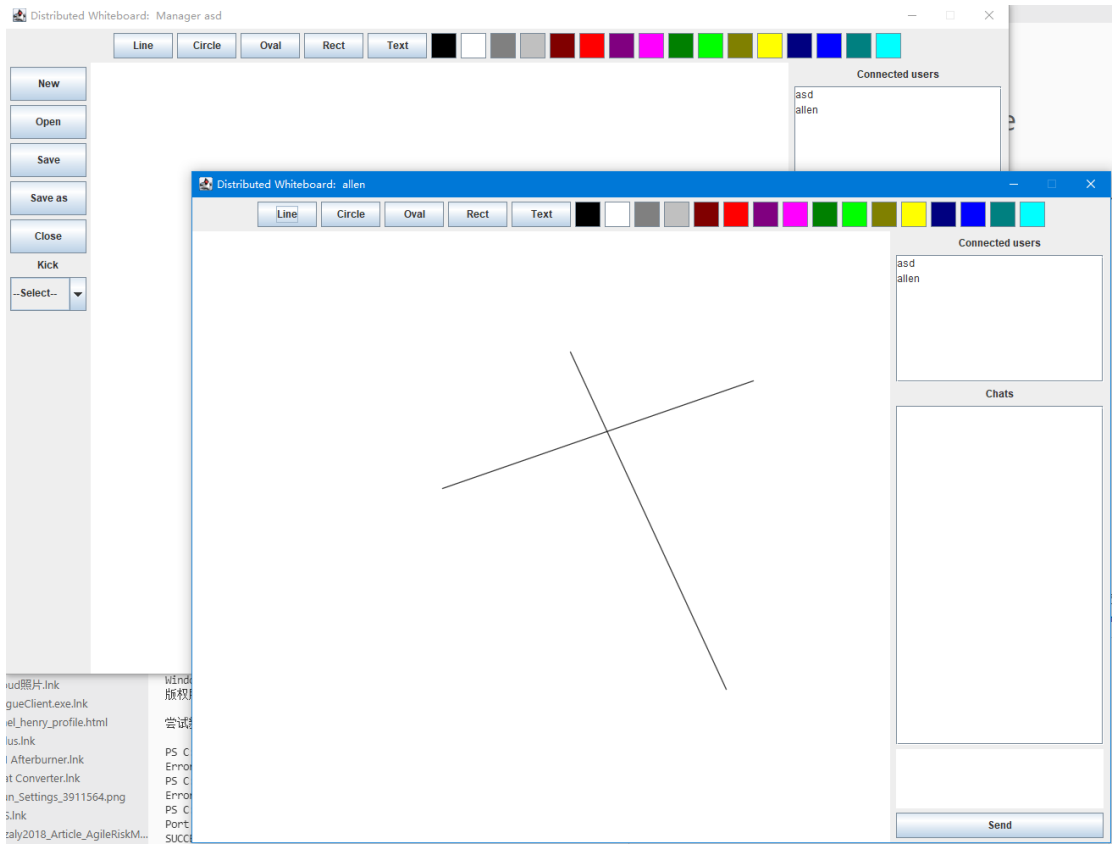
CONCURRENCY SOLVING

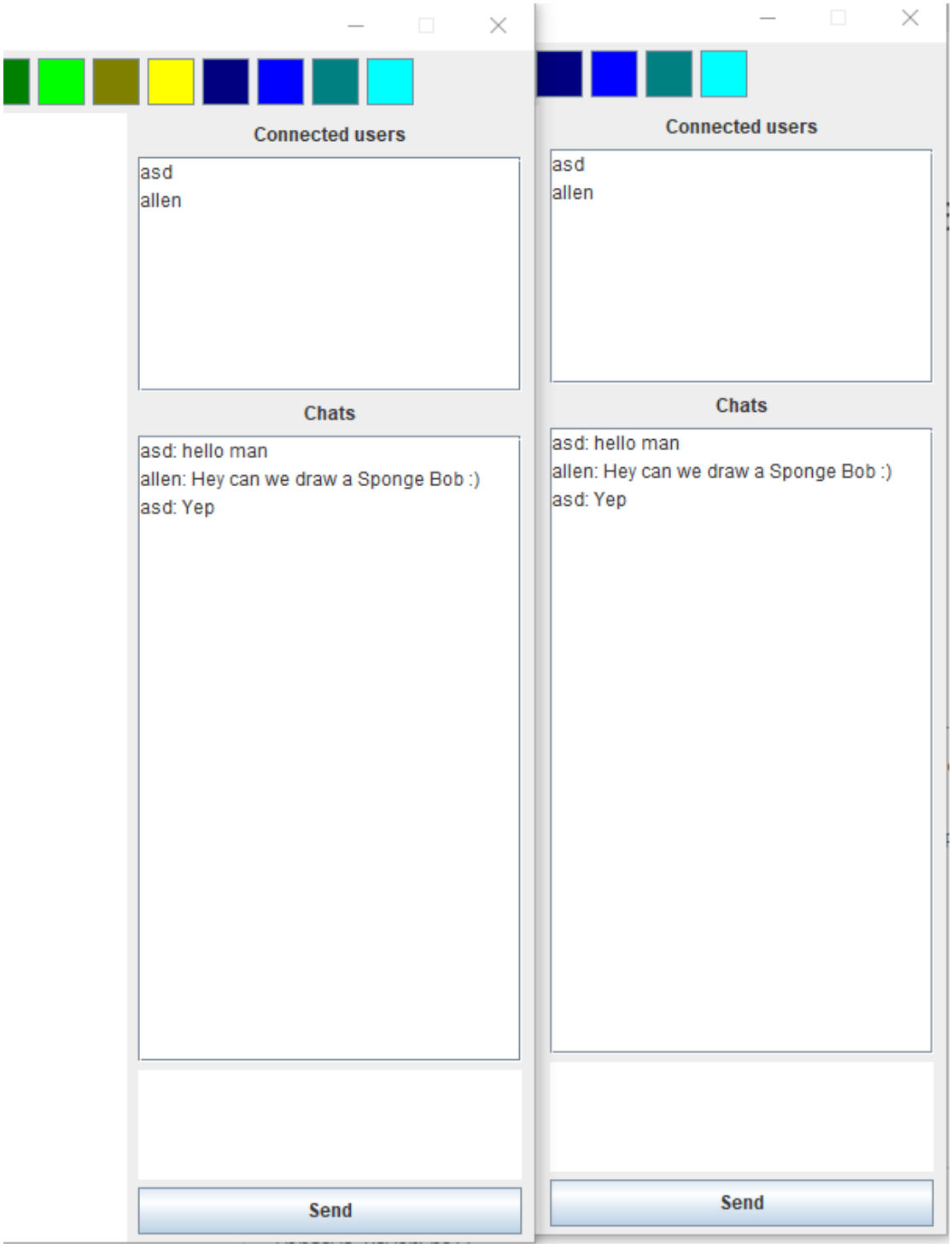
As mentioned in section of server architecture designing, all ServerThreads share the data stored in the Server class, and locks are added to those data, in the form of static synchronized getter and setter methods.

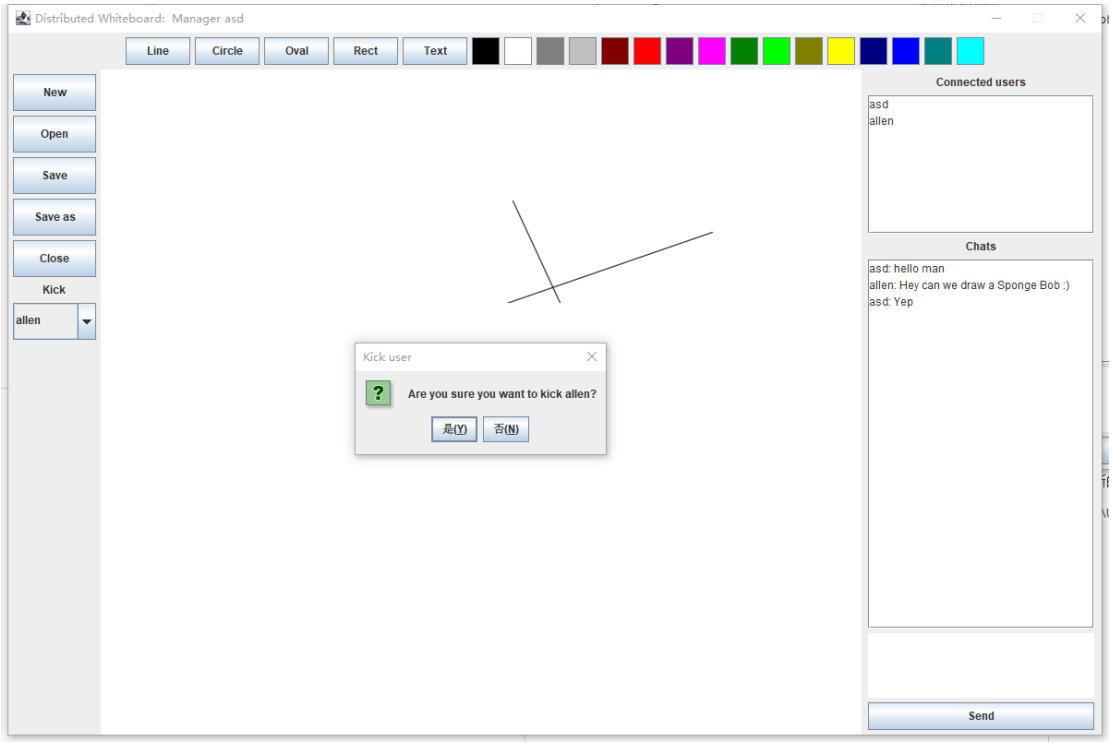
On the client side, there is no need for considering about concurrency issue. The Graphics2D object is created in the Listener class, along with an ArrayList for all shapes. However, Listener does not use these data directly, but letting ClientThread to do the job. This means there would be one singleton ClientThread manipulate with board and shapes only, which would only process messages one by one rather than letting the next message interrupts the currently processing one.

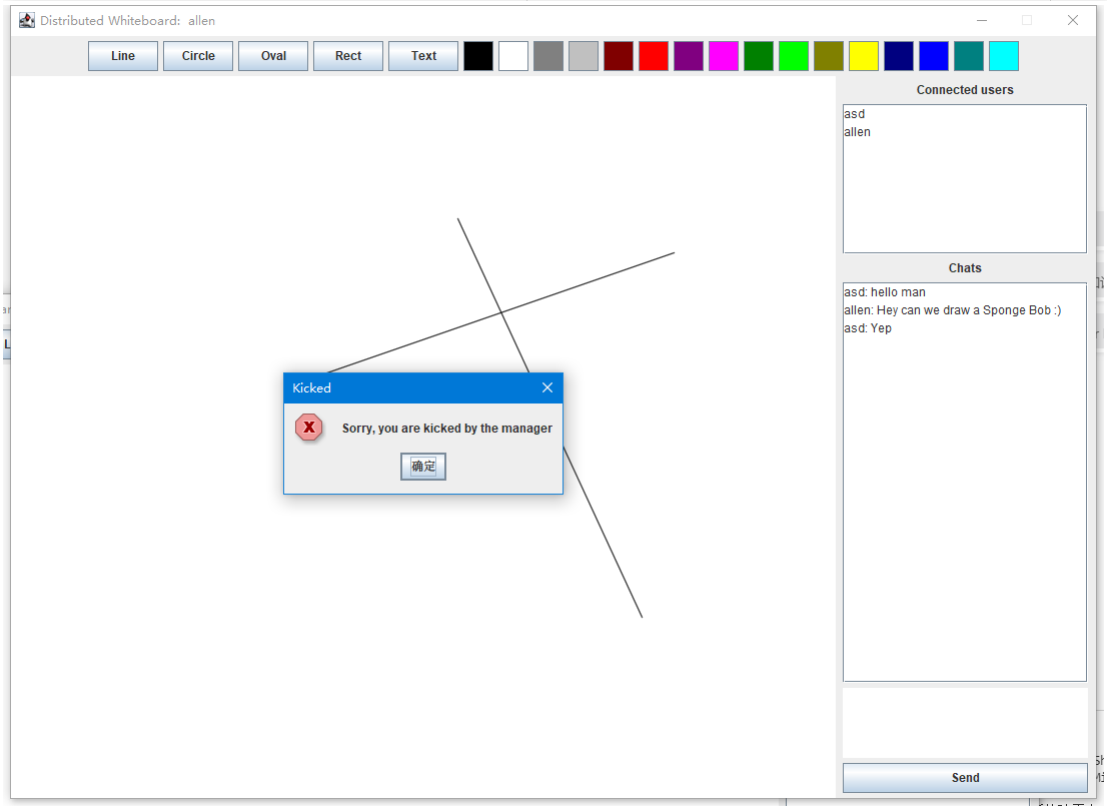
APPENDIX


Distributed Whiteboard: Manager asd
Line
Circle
Oval
Rect
Text
New
Open
Save
Save as
Close
Kick
--Select--
Connected users
asd
Chats
Send
New user
allen wants to join your whiteboard
是(Y)
否(N)


Distributed Whiteboard: Manager asd
Line
Circle
Oval
Rect
Text
New
Open
Save
Save as
Close
Kick
--Select--
Connected users
asd
allen
Distributed Whiteboard: allen
Line
Circle
Oval
Rect
Text
Connected users
asd
allen
Chats
Send


Connected users
asd
allen
Chats
asd: hello man
allen: Hey can we draw a Sponge Bob :)
asd: Yep
Send
Connected users
asd
allen
Chats
asd: hello man
allen: Hey can we draw a Sponge Bob :)
asd: Yep
Send


Distributed Whiteboard: Manager asd
Line
Circle
Oval
Rect
Text
New
Open
Save
Save as
Close
Kick
allen
Connected users
asd
allen
Chats
asd: hello man
allen: Hey can we draw a Sponge Bob :)
asd: Yep
Send
Kick user
Are you sure you want to kick allen?
是(Y)
否(N)


Distributed Whiteboard: allen
Line
Circle
Oval
Rect
Text
Connected users
asd
allen
Chats
asd: hello man
allen: Hey can we draw a Sponge Bob :)
asd: Yep
Send
Kicked
Sorry, you are kicked by the manager
确定