COMP90015: Distributed Systems – Assignment 2

Distributed Shared White Board

Yingnan Shi (yingnans1) 1025903

Siyu Bian (siyub) 984002

Qidi Wu (QIDIW) 978246

Zheng Tang (ztang3) 877030

1. Introduction

This project developed a shared white board that allows multiple users to draw simultaneously, which implemented a single server-clients architecture. It also has a function of instant chatting that allows users to broadcast messages to other users.

The project was based on socket and multiple threads to connect the server and clients and support multiple operations performed at the same time. More specifically, TCP sockets was chosen in this project in this project are based on the reliability of communication. And it implemented two threads for client to support drawing and chatting separately.

In this report, the system architecture is introduced, followed by the description of the communication protocols and message formats. After showing the design diagrams of both class and interaction, the conclusion of the system is briefly illustrated finally.

2. The system architecture

The single server-clients architecture is applied in this system, and the communication between the server and clients implements TCP sockets. There are 2 components designed in this project: server and client.

2.1 Server

The server program serves as an intermediary between among clients, which provides the information exchange services among multiple users: 1. receiving the drawing operations from the user and keeping all the active users updated with the current state of the whiteboard; 2. Listening to the chatting message from users and broadcasting to the other users; 3. storing the information of each active user and enabling only the manage user has the authority of some particular operations such as removing a user and closing the current canvas and so on.

2.2 Client

The client program will show users a shared graphical interface with two windows after successfully connecting to the server: WhiteBoard and Chat Window (see Figure 1). Users must provide a unique name when joining the whiteboard. In this system, the first user who creates the whiteboard is the manager. Whenever a new user joins the system, the user can do all the drawing operations as well as chat with others. The corresponding username will be added into the user list. Besides, when the manager quits, the application will be terminated, and all users will get a caution.

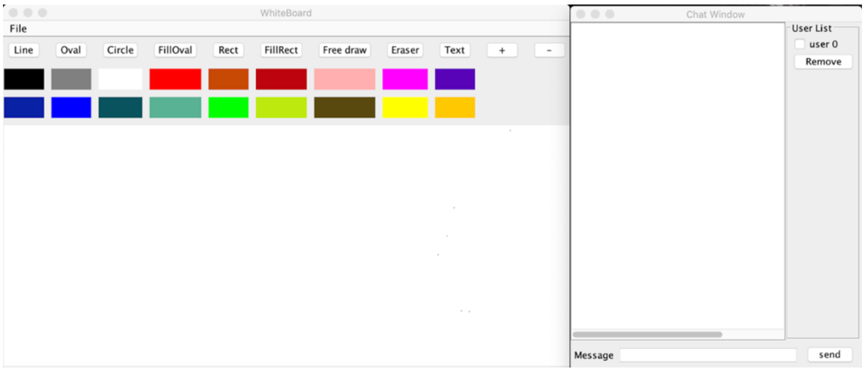


Figure 1. The Client Interface

2.2.1 WhiteBoard

The GridLayout is used to implement the WhiteBoard which can be shared between multiple users. All users can see the same image of the whiteboard with a short delay. They can draw line, oval, circle, rectangular as well as free drawing and erasing. Also, users can choose their favorite color and brush size to draw the above features. Text input is allowed everywhere inside the whiteboard with different font size. Besides, when someone is drawing, the username is shown in the WhiteBoard as its name.

The file menu which is located on the top left of the frame provides five functions including new, open, save, save as and close. However, the general user can use the save as function to save the current drawing as an image. The other four functions can only be implemented by the manager: 1. save: save the current drawing as a .dat file; 2. open: open a .dat file which stores the graph data, and the users can edit the figure drawn before; 3. new: clear the current drawing and create a new canvas. 4. close: clear the current whiteboard and close the drawing function.

2.2.2 Chat Window

The BorderLayout is used to implement the Chat Window. There are three sections:

- Chat content

A JTextArea contained in a JScrollPane to allow the previous messages can be seen.

- Send message

A “Message” Label, an input JTextField and a “send” JButton.

- User List

One JCheckBox representing one user.

When a user sent a message, the corresponding username followed by the message will be shown on the chat content section. A notification will be delivered to the manager if a new user applies to join in. Also, only the manager can remove someone at any time by clicking the button. The user list can be updated in real time after a new user joining in or users being removed.

3. Communication Protocols and Message Formats

The communication protocol between the server and the client is simply defined as a string. Different separators are used to transfer different kind of messages. By splitting the message, the server can response the requests of users, and then users can do the corresponding operations. Additionally, some updates and authorization also implemented by using communication protocols.

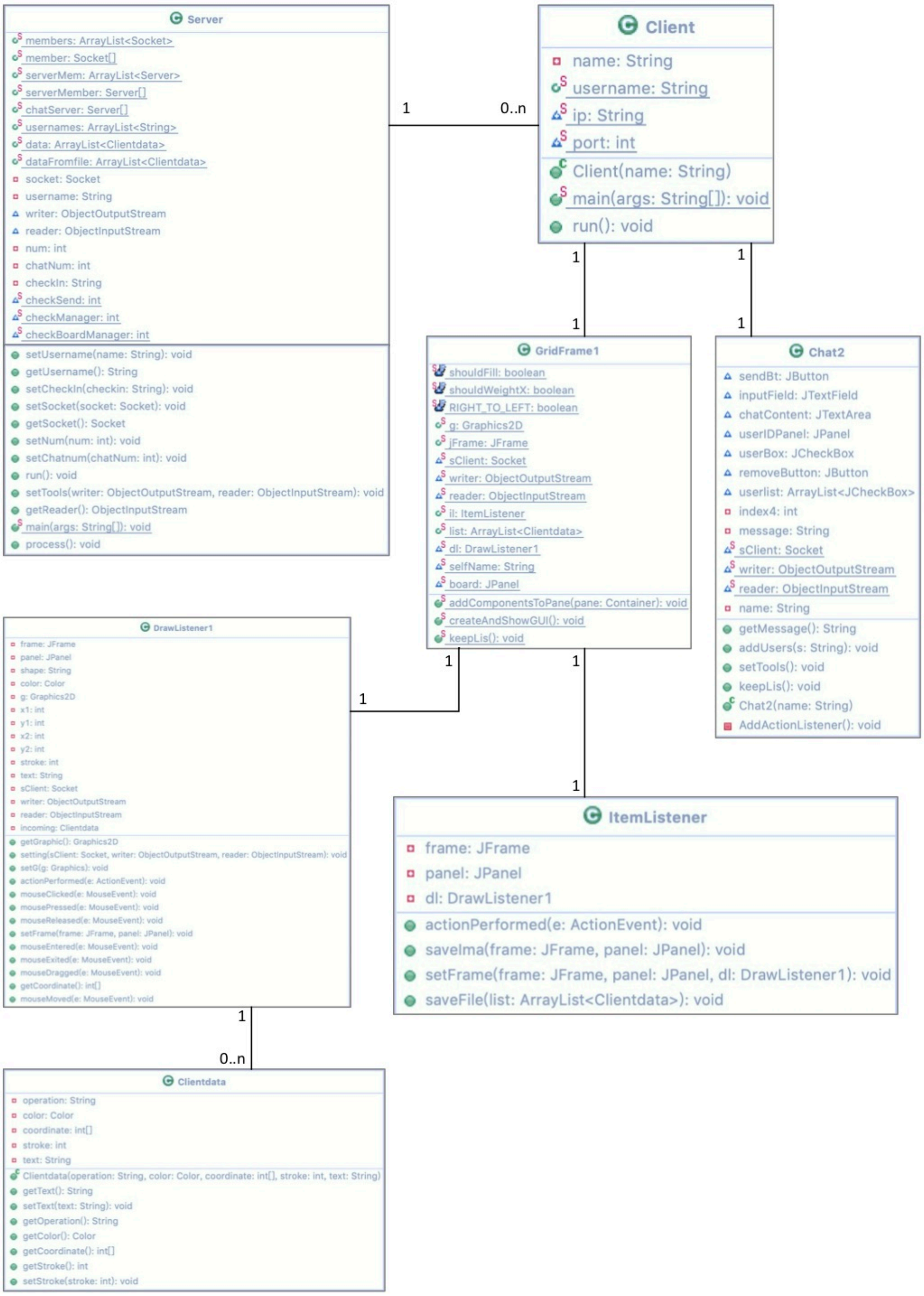
3.1 WhiteBoard

In this system, only the manager can do the new, open, save and close operations. We defined one unique message for each operation. If the server receives a request mentioned before from the manager, a specific message will be sent to the manager. This can ensure that only the manager can do these operations.

3.2 Chat

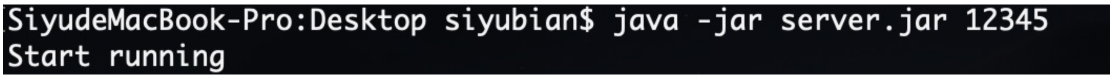
"@@" is used in the communication between the server and the Chat Window. The server appends the "username" and each username connected with "@@". This is used for updating the user list and showing the corresponding username when users send messages.

4. Design Diagrams (class and interaction)


Server
members: ArrayList<Socket>
member: Socket[]
serverMem: ArrayList<Server>
serverMember: Server[]
chatServer: Server[]
usernames: ArrayList<String>
data: ArrayList<Clientdata>
dataFromfile: ArrayList<Clientdata>
socket: Socket
username: String
writer: ObjectOutputStream
reader: ObjectInputStream
num: int
chatNum: int
checkIn: String
checkSend: int
checkManager: int
checkBoardManager: int
setUsername(name: String): void
getUsername(): String
setCheckIn(checkin: String): void
setSocket(socket: Socket): void
getSocket(): Socket
setNum(num: int): void
setChatnum(chatNum: int): void
run(): void
setTools(writer: ObjectOutputStream, reader: ObjectInputStream): void
getReader(): ObjectInputStream
main(args: String[]): void
process(): void
1
0..n
Client
name: String
username: String
ip: String
port: int
Client(name: String)
main(args: String[]): void
run(): void
1
1
1
1
GridFrame1
shouldFill: boolean
shouldWeightX: boolean
RIGHT_TO_LEFT: boolean
g: Graphics2D
jFrame: JFrame
sClient: Socket
writer: ObjectOutputStream
reader: ObjectInputStream
il: ItemListener
list: ArrayList<Clientdata>
dl: DrawListener1
selfName: String
board: JPanel
addComponentsToPane(pane: Container): void
createAndShowGUI(): void
keepLis(): void
Chat2
sendBt: JButton
inputField: JTextField
chatContent: JTextArea
userIDPanel: JPanel
userBox: JCheckBox
removeButton: JButton
userlist: ArrayList<JCheckBox>
index4: int
message: String
sClient: Socket
writer: ObjectOutputStream
reader: ObjectInputStream
name: String
getMessage(): String
addUsers(s: String): void
setTools(): void
keepLis(): void
Chat2(name: String)
AddActionListener(): void
1
1
1
DrawListener1
frame: JFrame
panel: JPanel
shape: String
color: Color
g: Graphics2D
x1: int
y1: int
x2: int
y2: int
stroke: int
text: String
sClient: Socket
writer: ObjectOutputStream
reader: ObjectInputStream
incoming: Clientdata
getGraphic(): Graphics2D
setting(sClient: Socket, writer: ObjectOutputStream, reader: ObjectInputStream): void
setG(g: Graphics): void
actionPerformed(e: ActionEvent): void
mouseClicked(e: MouseEvent): void
mousePressed(e: MouseEvent): void
mouseReleased(e: MouseEvent): void
setFrame(frame: JFrame, panel: JPanel): void
mouseEntered(e: MouseEvent): void
mouseExited(e: MouseEvent): void
mouseDragged(e: MouseEvent): void
getCoordinate(): int[]
mouseMoved(e: MouseEvent): void
1
ItemListener
frame: JFrame
panel: JPanel
dl: DrawListener1
actionPerformed(e: ActionEvent): void
savelma(frame: JFrame, panel: JPanel): void
setFrame(frame: JFrame, panel: JPanel, dl: DrawListener1): void
saveFile(list: ArrayList<Clientdata>): void
1
0..n
Clientdata
operation: String
color: Color
coordinate: int[]
stroke: int
text: String
Clientdata(operation: String, color: Color, coordinate: int[], stroke: int, text: String)
getText(): String
setText(text: String): void
getOperation(): String
getColor(): Color
getCoordinate(): int[]
getStroke(): int
setStroke(stroke: int): void

5. Implementation

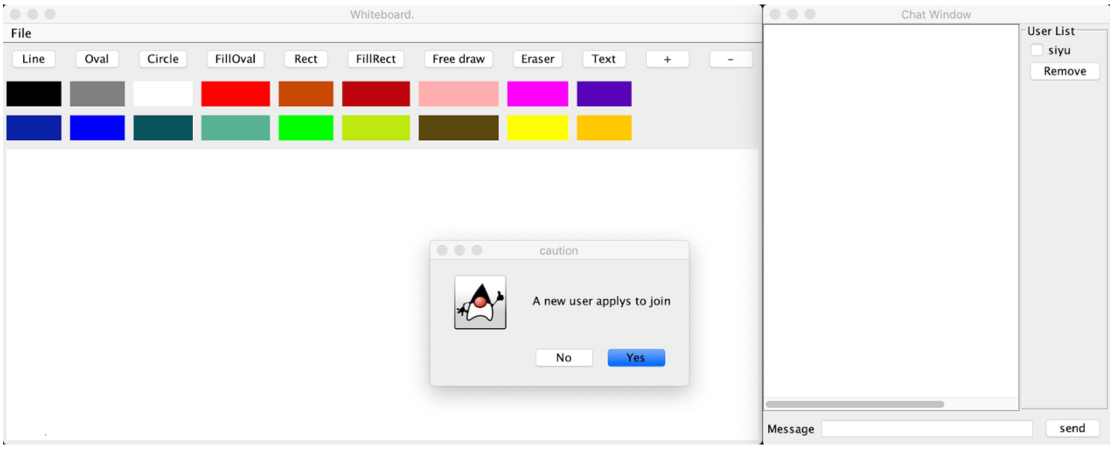
The .jar file and terminal are used to execute our program. The input format for server is "java -jar server.jar <Port>" (see Figure 2). "java -jar client.jar <IP> <Port> <Username>" is applied for users.



```
SiyudeMacBook-Pro:Desktop siyubian$ java -jar server.jar 12345
Start running
```

Figure 2

The first user who creates the whiteboard is the manager. When a new user apply to join in, a popup window will be shown for the manager to decide whether to allow the new user join in (see Figure 3).

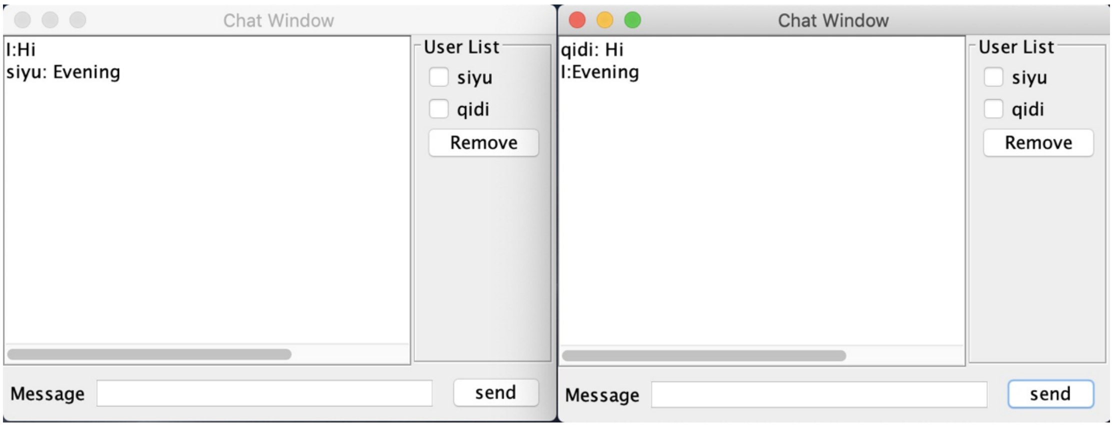


A new user applys to join

Figure 3

After the new user join in, the User List will be updated (see Figure 4).

When a user sent a message, "I" or the corresponding username will be shown followed by the input message (see Figure 4). The same image can also be seen in real time (see Figure 5).



I:Hi
siyu: Evening

qidi: Hi
I:Evening

Figure 4

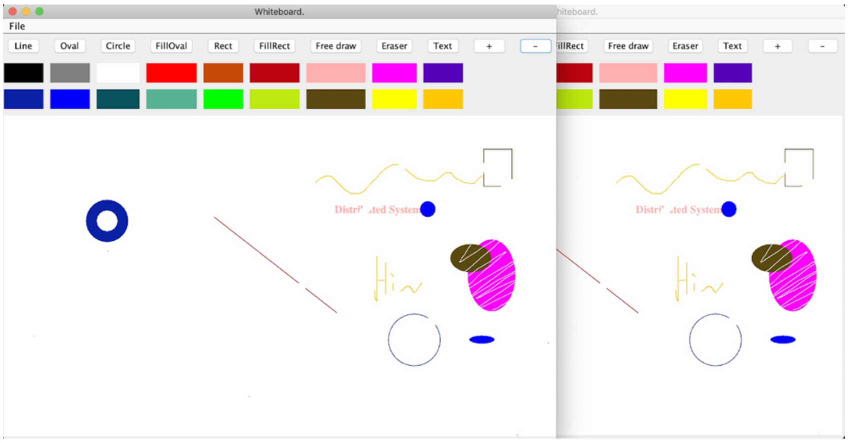


Figure 5

6. New Innovations

6.1 GridBagLayout

Compared with BorderLayout, the default layout, GridBagLayout is more flexible. All components can be scaled when users resize the window (see Figure 6). However, it is difficult to adjust the position and size of components, because of its nature.

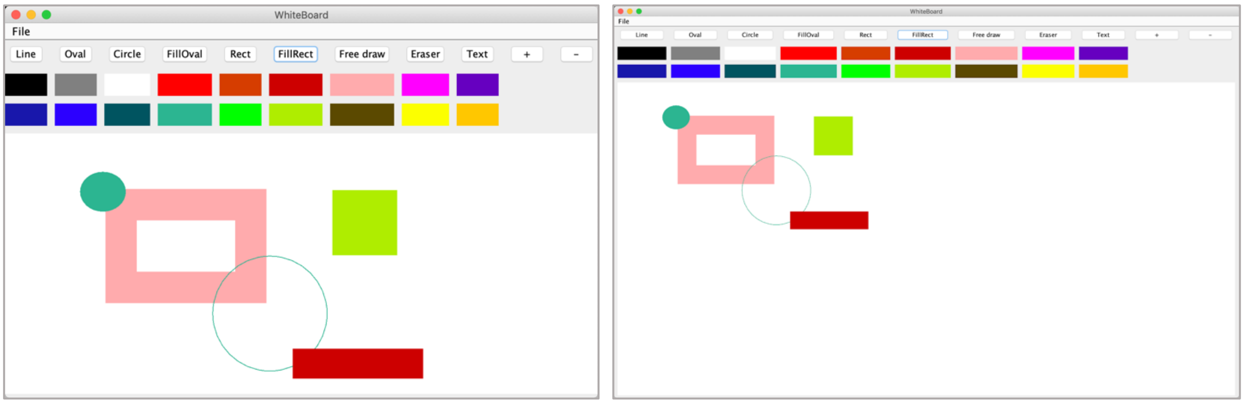


Figure 6. Resize the Window

6.2 Filled Shapes

Except drawing normal graphs, for closed graphics (oval and rectangular), color filling is implemented, even though the color of the border cannot be changed (see Figure 6).

7. Outlines the Contribution

Name & Std No,	Contribution Area	Overall Contribution
Yingnan Shi & 1025903	All the rest of the work were done evenly (Socket/thread, data exchange, GUI, Chat…)	30%
Siyu Bian & 984002		30%
Qidi Wu & 978246		30%
Zheng Tang & 877030	buttons in Menu except "saveAs" in Phase 1(which has no interaction with server)	10%