Distributed Systems
Assignment 2 Report

Qingfeng Xu 967175
Huan Cao 985151
Dongfang Wang 906257

Contents

1 Introduction

This project develops a shared whiteboard which allows more than one user work on a canvas at the same time. With this software, different clients could work share the work that they have done on their own board. As drawers, they can draw lines or shapes. Also, they can add text box on the draw box. The server will maintain its current state and manage the operations of all clients. In addition, there are some other special features, such as user chat, server backup.

2 Design

2.1 Architecture

2.1.1 Protocol

- **TCP:** The sockets are **TCP**. Because the TCP is connection oriented and reliable, all the communications of this system are reliable.
- **Socket based:** All communications of the system are based on **socket**.
- **Server and client setting:** The configure of the server and client number are dominated by jar command. The default is 8080.
- This system is based on a **client-server**.

2.1.2 Thread Management

The multi-threaded server implements a **thread-per-connection**. The CreateWhiteBoard class will create the server with the port number and username. Then each client created by JoinWhiteBoard class will connect with server. The server will create a thread for each connection. All the threads will be maintained in a thread list by server. For each connection, we create a receiving message thread to receive message. The receiving message thread will not influence the main thread of client and server. (Figure 2.1)

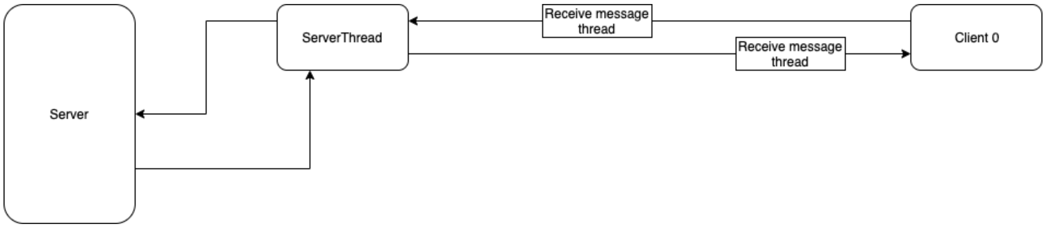


Figure 2.1 Thread Management

When a new client need to join, it must ask the **permission from server**, firstly. (Figure 2.2) Once a client joins, it must provide a username. The server also distributes a unique client number to each user. To avoid the same name of user, we use the combination of username and client number to identify a user. The server has the right to **disconnect with the client**. (Figure 2.3)

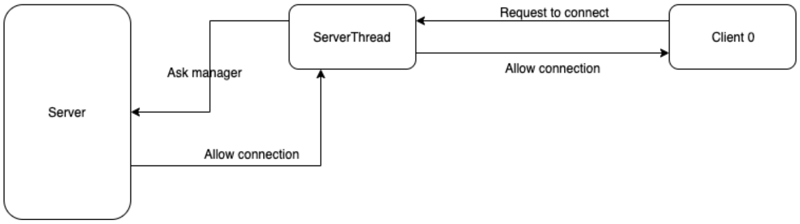


Figure 2.2 The Connection diagram

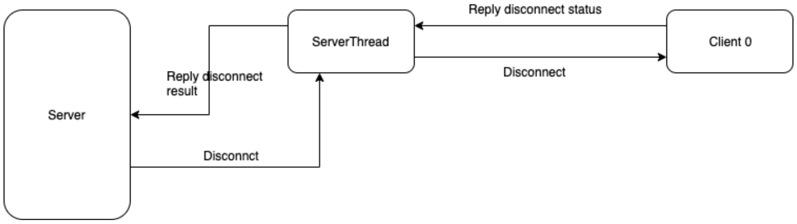


Figure 2.3 The Disconnection diagram

2.2 Interaction

The diagram below is the general interaction diagram. (Figure 2.4) If server has any update, it will broadcast to all clients. While clients have update, it will send to server at first, then the server broadcast to all clients.

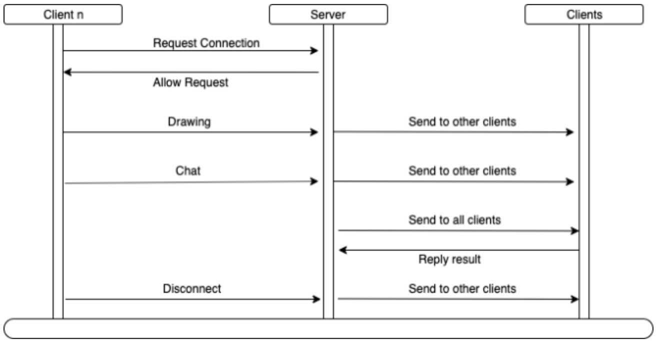


Figure 2.4 General Interaction diagram

2.2.1 Server Command Interaction Diagram

Figure 2.5 shows the interaction of server command (New, Open, Close). The server obtains the command message from the server GUI. Then, the server sends command to each client through threads list. The client receives the command and executes the command. After the command has been applied on client window, then it will feedback the results to server. Then, server can update the status of clients.

Data Format: JSON file is used to store the message

```
{
“Command”: “command”
“Content”: “New File”
}
```

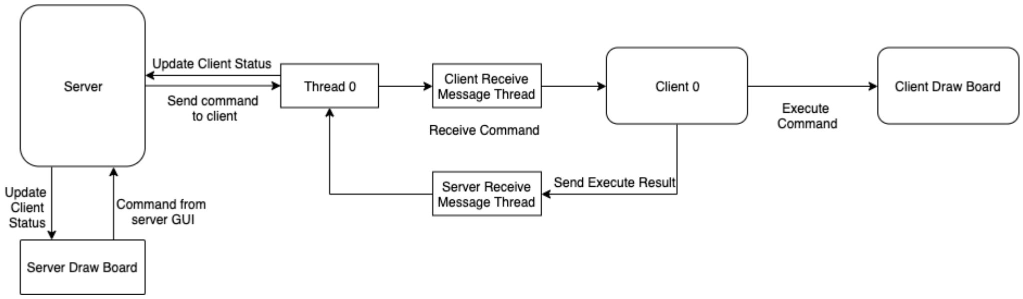


Figure 2.5 Server Command Interaction Diagram

2.2.2 Drawing Interaction

Figure 2.6 demonstrates the communication diagram that update drawing action from server. One the server get the drawing action from server GUI, then the server will send the drawing action to all clients. After clients receiving the drawing action from server, the client will update the drawing action on their own drawing board.

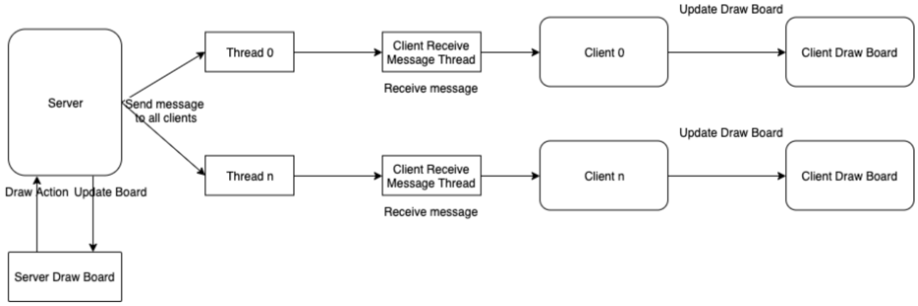


Figure 2.6 The diagram of update drawing action from server

Figure 2.7 demonstrates the communication diagram that update drawing action from client. Because the client cannot send message to other clients directly. It will send the message to server, firstly. Then, the server will send the drawing action to all clients. After clients receiving the drawing action from server, the client will update the drawing action on their own drawing board.

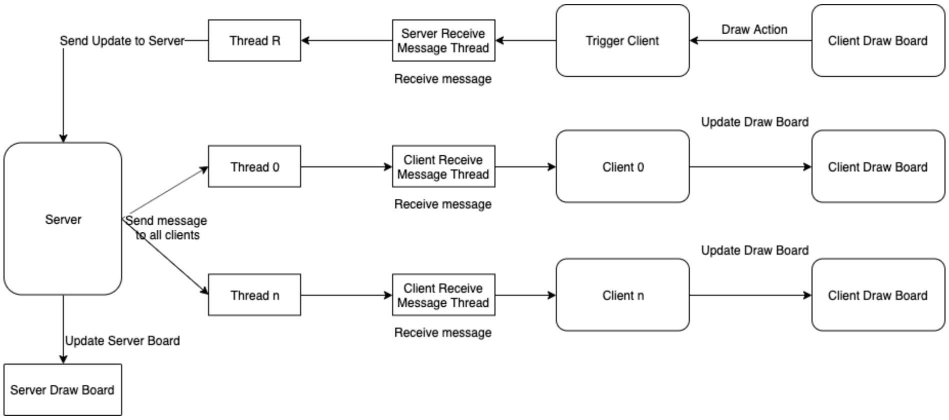


Figure 2.7 The diagram of update drawing action from client

Data Format: JSON file is used to store the message

```
{
"command": draw
"shapeName": circle
"shapeText": none
"shapeColor": black
"shapeInitialX":10
"shapeInitialY":11
"shapeFinalX":40
"shapeFinalY":40
"shapeWidth":5
}
```

2.2.3 Chat Interaction

Figure 2.8 shows the diagram of chat window. Once the user sends one chat message. It will send to server directly. Then the server will send this message to all client windows.

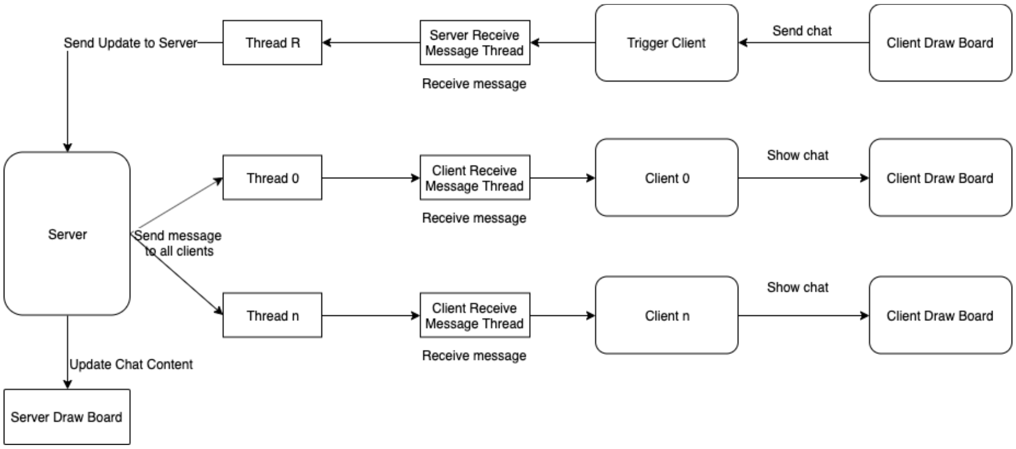


Figure 2.8 The diagram of chat

Interaction Detail:

The parameter from the client input is extracted and it is also sent to the server via the socket. After the server receives it, the server sends it to each client through the socket and displays it in the chat record.

Data Format: JSON file is used to store the message

```
{
"command": chat
"text": this is chat message
"time": 10:22:00
"clientNumber":10
}
```

2.3 Class design

The UML is shown in Appendix

- **CreateWhiteBoard:** This is the class to create the server and server window. The server and server window also will be connected in this class.
- **Server:** This is the class that to create the TCP server with listening connection request from client. Once a connection build, it will create a server thread to maintain this connection. The methods that send message to all clients also are included in this part.
- **ServerThread:** This is the class that implements Runnable to add a new thread to keep the connection with client. The information of client such as username and client number are included in this class.
- **ListenConnectionThread:** The server uses this thread to create a connection
- **ShutDownWork:** This thread is the thread that executed when the server is closed. The socket is closed and file is saved automatically.
- **JoinWhiteBoard:** This is the class to create the client and client window. The client and client window also will be connected in this class.
- **Client:** This is the class to create the TCP client. It will ask the permission of connect with server. Then, it will create TCP connection with server.
- **ClientShutdownWork:** This is the class that executed when the client is exiting. It will send the exit information of the client.
- **WhiteBoardGUI:** This is the class that show the GUI window. Both server and client use the same style of GUI. However, the client has no authority to some function: Manage Client, New, Save, Open, Close.
- **Message:** This is the class to format the message that need to send
- **DrawMethod:** This is the class to draw shapes and stores the information of shapes.
- **ShapeList:** This is the class that store the shape information, including name, color, width, position.
- **OpenFile:** This is the class to read shapes from file.
- **SaveFile:** This is the class to save files or save as files.
- **Log:** This is the class to record the drawing action of each user.

2.4 GUI

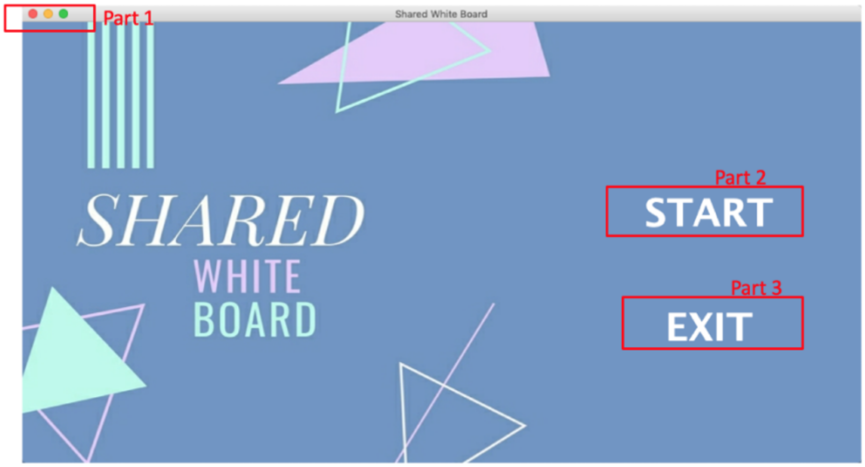


Figure 2.9. Start Interface

The server and the client have the same GUI interface.

Figure 2.9 shows the start page of the drawing board.

- Part1: Close Program Button

If this button is clicked, the program will prompt that “Do you want to exit the Shared White Board?”. If so, the program will save the content of the white board and close it.

- Part2: START Button

If this button is clicked, the program will jump to the Shared White Board interface.

- Part3: EXIT Button

If this button is clicked, the program will prompt that “Do you want to exit the Shared White Board?”. If so, the program will save the content of the white board and close it.

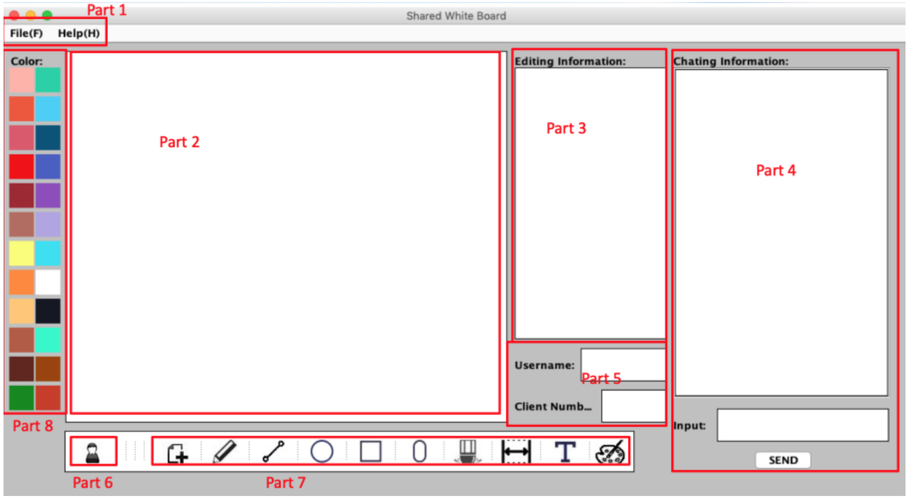


Figure 2.10. Drawing board home page

Figure 2.10 Shows the drawing board home page.

- Part 1: File management Menu.

Including New, Open, Save, Close and Save as functionality. The client only is only allowed to use save as to save picture in local. The server can use all functions.

- Part 2: Draw Board.

This is the space to draw the shapes or add text box.

- Part 3: Editing Clients.

For server, it will show the client’s drawing action with client number and time. For client, it will show the client’s name who are editing the drawing board.

- Part 4: Showing chat information.

This is the part to show the chat history. Also, the user can send message here.

- Part 5: Showing the client information.

This is the part to show the client’s username and client’s number.

- Part 6: Client management.

Only the server has the right to manage the client. It shows the cilent’s information. And the manager can input the client number that he wants to disconnect and disconnect with it.

- Part 7: Draw tools.

The user can select drawing free line, line, circle and other shapes. Users also can add text box here. Moreover, the client can select the width and color by selecting default size or input configuration by themselves.

- Part 8: Select color.

The Drawing board provide 20 colors for user to set color easily.

2.5 Concurrency Management

To avoid concurrency, we take three operations on server: **Split listen drawing action and drawing on board**, which means we use listener to obtain the shape that somebody who are drawing, after all information collected (shape, location, color, width), we generate drawing task and draw on board. By this means, the user's drawing action will not conflict with the update of board from other clients. Meanwhile, **we lock the board panel when it executes a drawing command.** Therefore, the other process will not influence the editing task. Thirdly, **if there are two or more update commands arriving at the same time, the window will maintain a task list which stores all tasks.** The order is according to the client number. The smaller the client number, the higher priority the client has. Figure 2.11 shows the logic that server manage the concurrency.

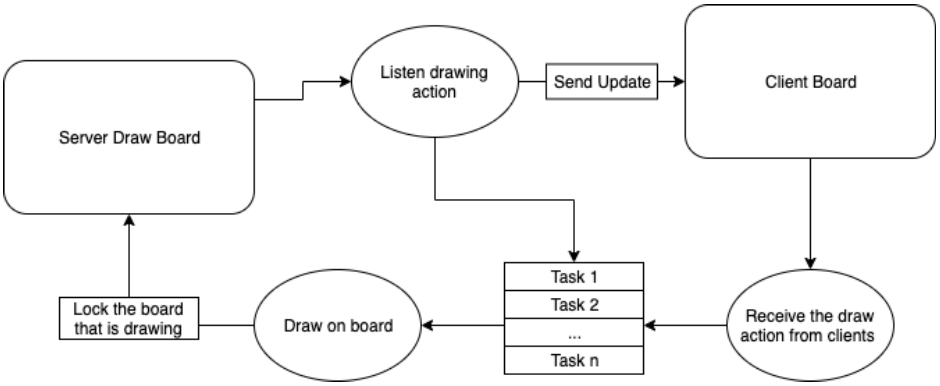


Figure 2.11 The logic of server to manage the concurrency

From 2.11, the client only receive the message from client. Even though other clients need to update something, it also will send to server, firstly. Therefore, the only concurrency for the client is tasks from server with client's operation. We use the **similar concurrency handling method to server**. But the server only need to handle the task order according to the **order that server send to client.**

3 Functionality

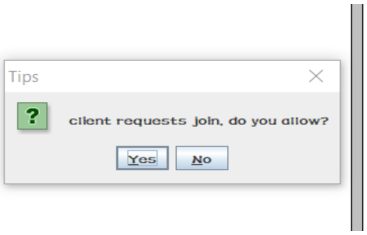
3.1 Manager

3.1.1 Create manager

Manager window is created by the class: CreatWhiteBoard, and the manager needs to write three arguments: IP address, port number and username.

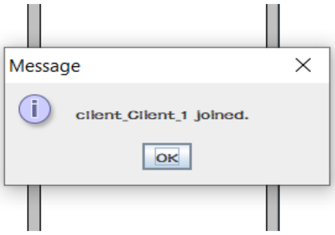
3.1.2 Allow to join

If a client wants to join in the whiteboard, the manager can receive a requirement to determine whether allow this client to join the whiteboard. It will prompt that “client requests join, do you allow?” in manager interface. (Figure 3.1) If the manager allows to join, it will also prompt that “client joined” in manager interface. (Figure 3.2)



client requests join, do you allow?

Figure 3.1 Manager Interface

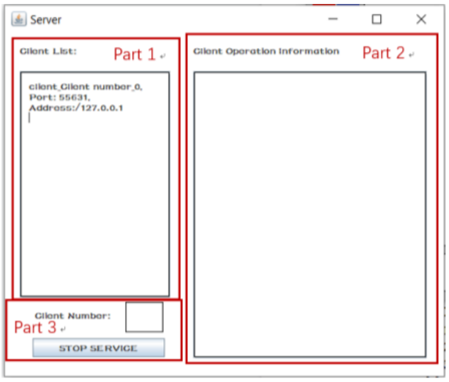


client_Client_1 joined.

Figure 3.2 Client Interface

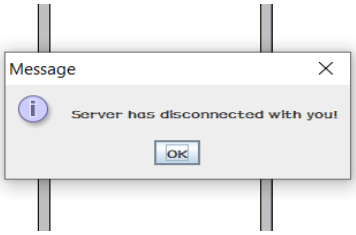
3.1.3 Manage Clients

The manager can manage clients. There are parts in manage interface, and only the manager can open this interface. (Figure 3.3)



client_Client number_0,
Port: 55631,
Address:/127.0.0.1

Figure 3.3 Manage Client Interface



Server has disconnected with you!

Figure 3.4 Client Interface

Part1: Client List Part

This part record the client information, such as client username, client port number and client IP address.

Part2: Client Operation Information Part

This part record the client operation information.

Part3: Disconnect Client Part

This part is used to disconnect a client. The manager can input the client number which he wants to disconnect and click the stop button, the client will disconnect with the whiteboard. Then, the client will receive a prompt “Server has disconnected with you!” and exit the system. (Figure 3.4)

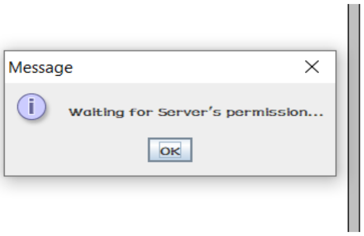
3.2 Client

3.2.1 Create client

Client window is created by the class: JoinWhiteBoard, and the client needs to write three arguments: IP address, port number and username.

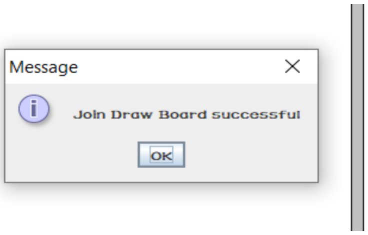
3.2.2 Apply to join

If a client wants to join in the whiteboard, he will receive a prompt "Waiting for Server's permission...". (Figure 3.5) If the manager allows him to join, he will receive a prompt "Join draw board successfully". (Figure 3.6) If not, he will receive a prompt "Server has disconnected with you!". (Figure 3.7)


Message

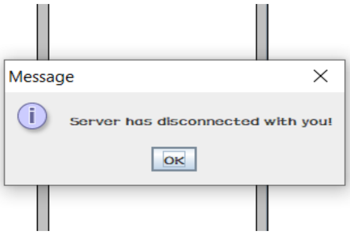
Waiting for Server's permission...

Figure 3.5 Client Interface


Message

Join Draw Board successful

Figure 3.6 Manager Interface


Message

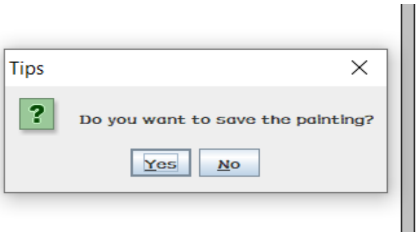
Server has disconnected with you!

Figure 3.7 Client Interface

3.3 Basic Operation

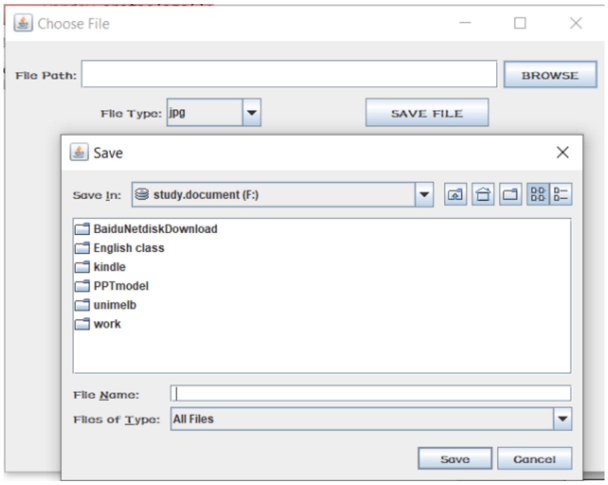
3.3.1 New

Only the manager can create a new file. If the manager clicks new menuitem in file menu, the system will prompt "Do you want to save the painting?". (Figure 3.8) If so, the system will save the painting at first. If not, the system will only create a new file.


Tips

Do you want to save the painting?

Figure 3.8 Save File


Choose File

File Path:

File Type: jpg

Save

Save In: study.document (F:)

BaiduNetdiskDownload
English class
kindle
PPTmodel
unimelb
work

File Name:

Files of Type: All Files

Figure 3.9 Save File Chooser

3.3.2 Save

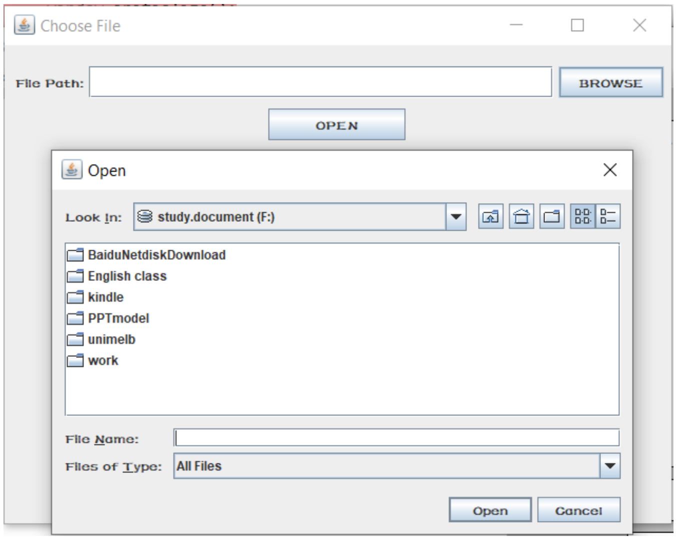
Only the manager can save a file. If the manager clicks save menuitem in file menu, the system will show a file choose frame and the manager needs to choose a file path to save the painting. Only the first time, the manager needs to choose a file path. (Figure 3.9) Then, if the manager wants to save again and again, the system will consider this file path as the default file path.

3.3.3 Save As

Both the manager and the client can save as a file. If the manager or the client clicks save as menuitem in file menu, the system will show a file choose frame and he needs to choose a file path to save the painting. (Figure 3.9) Each time he saves as painting, he always needs to choose a file path.

3.3.4 Open

Only the manager can open a file.If the manager click open menuitem in file menu, the system will prompt "Do you want to save the painting?". (Figure 3.8) If so, the system will save the painting at first. If not, the system will show a file choose frame and the manager needs to choose the file path to open a file. (Figure 3.10)



Look In: study.document (F:)

Files of Type: All Files

Figure 3.10 Open File Chooser

3.3.5 Close

Only the manager can close a file. If the manager clicks close menuitem in file menu, the system will prompt "Do you want to save the painting?". (Figure 3.8) If so, the system will save the painting at first. If not, the system will close the file.

3.4 Drawing

3.4.1 Draw Shape

Both of server and client can draw different shapes on the whiteboard. They can choose different color and width of the shapes, as well as fill or not. And we provide a preview effect with the mouse drag so that the user can better know the shape they draw. (Figure 3.11)

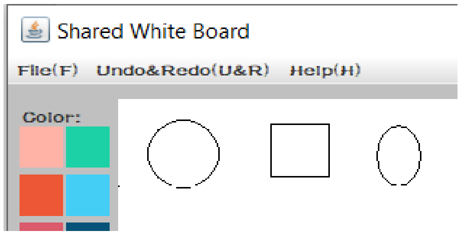


Figure 3.11 draw shape

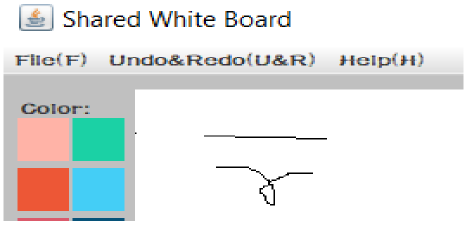


Figure 3.12 draw line and freedraw

3.4.2 Draw Line and Freedraw

Server and client can also draw line on the whiteboard. In addition to the most common lines, we also offer a variety of line drawing, such as dashed lines, arrows, etc. (Figure 3.12)

3.4.3 Text Box

In addition to drawing, we can also add text boxes. We can choose different width of the text boxes, different size and font of the text. (Figure 3.13)

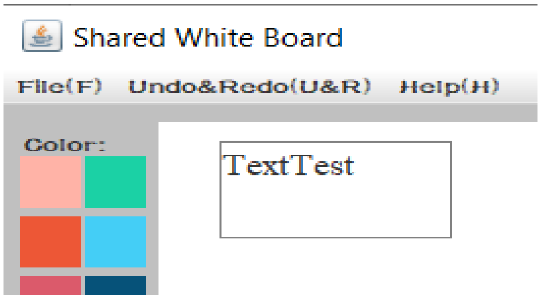


Figure 3.13 text box

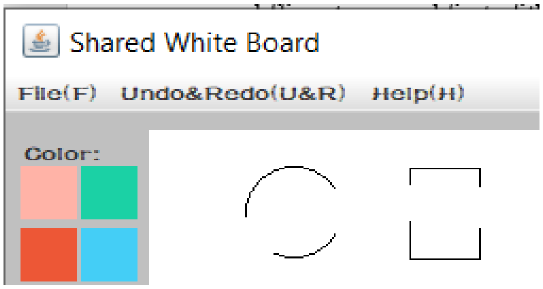


Figure 3.14 eraser

3.4.4 Eraser

We can use eraser to erase the drawn graphics, provide multiple width options. (Figure 3.14)

3.4.5 Width Adjustment

We can choose different width of shape by right clicking on the icon, if there is no suitable width, we can also input the specific width parameter to find the best width. (Figure 3.15, 3.16)

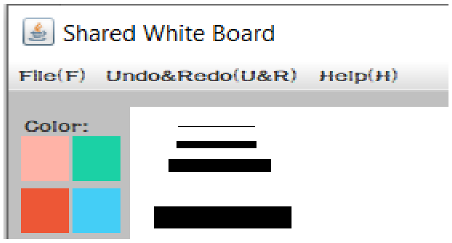


Figure 3.15 width adjustment

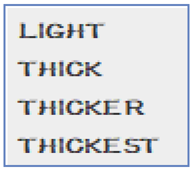


Figure 3.16 width adjustment

3.4.6 Color Adjustment

We can choose the different color of shape quickly on the left of whiteboard, if there is no suitable color, we can also choose more color by clicking the color icon. (Figure 3.17, 3.18)

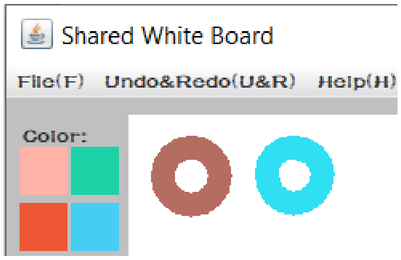


Figure 3.17 color adjustment

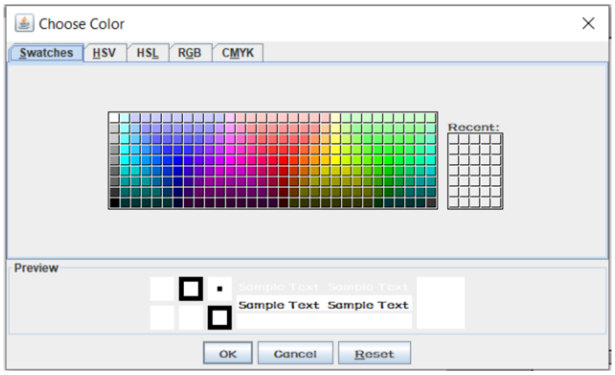
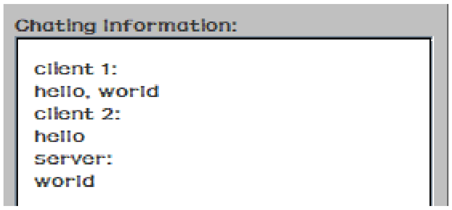


Figure 3.18 color adjustment

3.5 Chat

3.5.1 Normal Chat

Each client and the server can chat on the right of the whiteboard, the message will show the time, the username and the chat content. (Figure 3.19, 3.20)



client 1:
hello, world
client 2:
hello
server:
world

Figure 3.19 chat information record

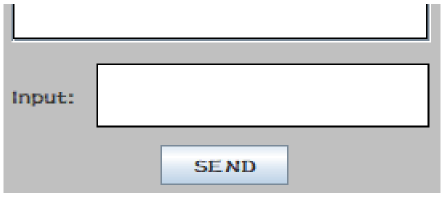


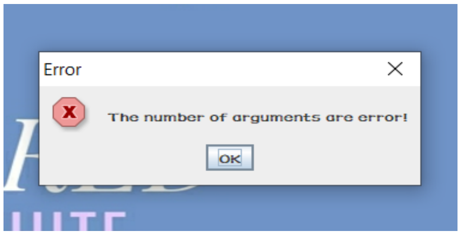
Figure 3.20 chat input

4 Exception Handling

4.1 Argument Exception

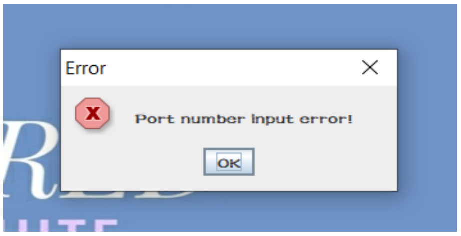
4.1.1 The number of arguments

Both the server and client have argument exception: the number of arguments. If the number of arguments is not equal to 3, the system will prompt that "The number of arguments are error!". (Figure 4.1)



The number of arguments are error!

Figure 4.1 the number of arguments error



Port number input error!

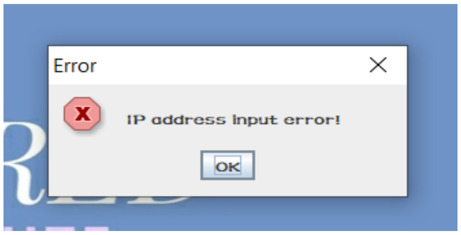
Figure 4.2 port number error

4.1.2 Port number

Both the server and client have argument exception: port number. If the port number is wrong, the system will prompt that “Port number input error!”. (Figure 4.2)

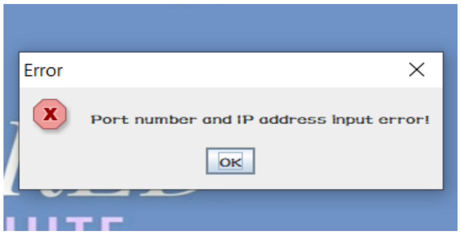
4.1.3 IP address

Both the server and client have argument exception: IP address. If the IP address is wrong, the system will prompt that “IP address input error!”. (Figure 4.3)



IP address input error!

Figure 4.3 IP address error



Port number and IP address input error!

Figure 4.4 IP address and port number error

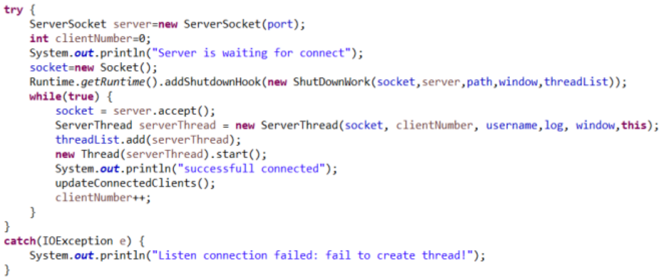
4.1.4 IP address and port number

Both the server and client have argument exception: IP address and port number. If the IP address and port number are all wrong, the system will prompt that “Port number and IP address input error!”. (Figure 4.4)

4.2 Connection Exception

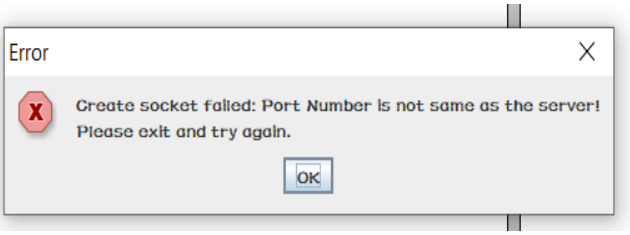
4.2.1 Server

If the thread occurs create error, the system will prompt that “Listen connection failed: fail to create thread!”. (Figure 4.5)



```
try {
    ServerSocket server=new ServerSocket(port);
    int clientNumber=0;
    System.out.println("Server is waiting for connect");
    socket=new Socket();
    Runtime.getRuntime().addShutdownHook(new ShutDownWork(socket,server,path,window,threadList));
    while(true) {
        socket = server.accept();
        ServerThread serverThread = new ServerThread(socket, clientNumber, username,log, window,this);
        threadList.add(serverThread);
        new Thread(serverThread).start();
        System.out.println("successfull connected");
        updateConnectedClients();
        clientNumber++;
    }
}
catch(IOException e) {
    System.out.println("Listen connection failed: fail to create thread!");
}
```

Figure 4.5 Listen connection failed: fail to create thread!



Create socket failed: Port Number is not same as the server!
Please exit and try again.

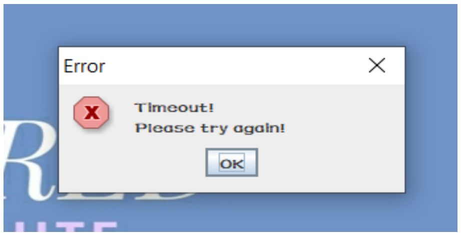
Figure 4.6 Client not same port number

4.2.2 Client

If the client and the server have different port number, the system will prompt that "Create socket failed: Port number is not same as the server! Please exit and try again!". (Figure 4.6)

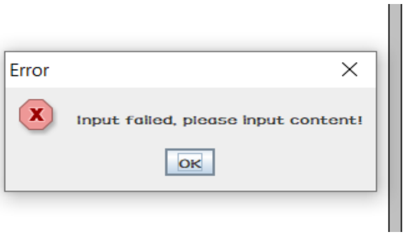
4.2.3 Timeout

If there is no client for a long time, the system will prompt that "Timeout! Please try again". (Figure 4.7)



Timeout!
Please try again!

Figure 4.7 Timeout error



Input failed, please input content!

Figure 4.8 Chat error

4.3 Chat Exception

If the chat input is empty, the system will prompt that "Input failed, please input content!". (Figure 4.8)

4.4 GUI Picture Path Exception

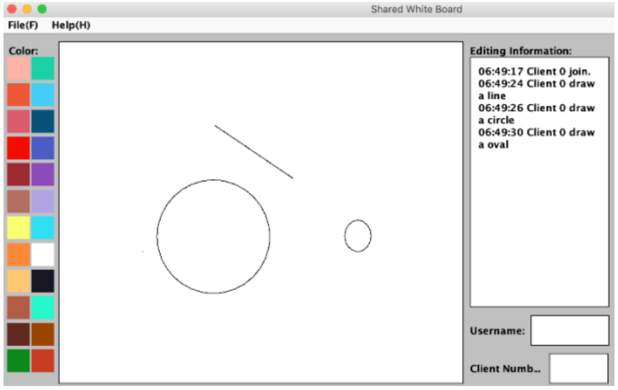
The command will prompt: "Error path!"

4.5 Other Exception

The command will prompt: InputStream, OutputStream, PrintWriter, BufferedReader I/O: "I/O error!"

5 Creative Element

5.1 Log File



```
06:49:17 Client 0 join.
06:49:24 Client 0 draw
a line
06:49:26 Client 0 draw
a circle
06:49:30 Client 0 draw
a oval
```

Figure 5.1 Server can read Log

To be more stable and safer, the server has log function. There are three kinds of information will be recorded: Server work information, message between server and clients, and the error message. Therefore, the manager can search the communication history and find the potential error. Moreover, even if the system broken down, the manager still can find the error message in log file and improve server program. The log window shows the log since the new server is built, while the log file stores all history log. The operation of each client will be recorded, including join board, leave board and draw something. The manager could know which client do what operation to manage the client better. Also, only the server has the right to see this log.

5.2 System Break Management

This is use hook thread which will be invoked when system shut down.

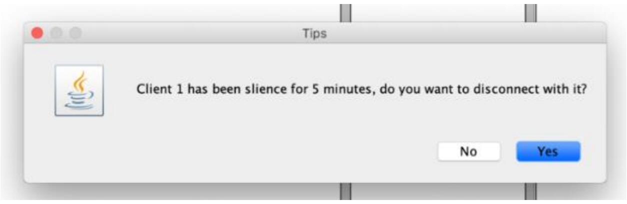
```
Runtime.getRuntime().addShutdownHook(new ShutDownWork(socket,server,loger));
```

The socket and server will be closed safely. Log will be written to the log file. Even if the server is broken down by accident, the system also can exit safely and record log, so that the manager can detect the accident by log file.

The file also be saved automatically, the default saving file name is "exceptionSaving.txt"

5.3 Auto Thread Management

Server disconnects with client that doesn't send message for a long time automatically: The client that is inactive could be seen as "Dead Client". Therefore, the server could disconnect with it to release resources. The server's son thread receives message by another thread ReceiveMessageThread and use Future to monitor the grandson thread. If this thread doesn't finish in 5 minutes, the server knows that the client didn't send and message in the past 5 minutes. Therefore, the connection will be broken in the Timeout Exception and the thread will be closed and deleted in server. (Figure 5.2)



Client 1 has been slience for 5 minutes, do you want to disconnect with it?

Figure 5.2 Ask disconnection with un-active client

5.4 Text Box Management

The font in text box has three font styles: "Time New Roman", "Calibri", "Chalkboard". In addition, the font can be bold and italicized.

5.5 Shape Style Management

The shape in the system, such as circle, rectangle and oval, can be filled. (Figure 5.3)

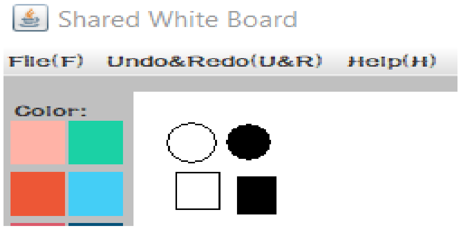


Figure 5.3 fill shape

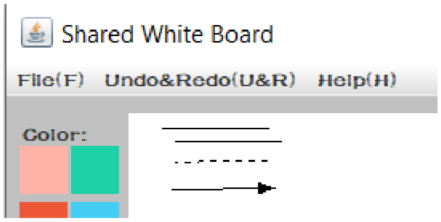


Figure 5.4 line style

5.6 Line Style Management

The line can be link, dashed line and directional connector. (Figure 5.4)

5.7 Choose The Shape

The system can use chooseShape button to choose the shape, and the shape can be moved by “w”-up, “a”-left, “s”-down and “d”-right. The shape which is chosen also can be resized. In addition, it can be deleted by “backspace” button on keyboard.

5.8 Redo And Undo

The system can redo or undo the previous operations using redo and undo menu.

6 Excellence

6.1 Consistence

- As described in Section 4, the system has a corresponding solution for each exception. Therefore, the system generally does not have an exception.
- To prevent the system from locking when the client is operating at the same time, all functions for changing data have been synchronized.
- Both server and client create the ShutDownWork child thread to close the system safely. Even if the system breaks down by accident, the operations in the whiteboard also will be stored.
- Because the server can control the client's permissions to change the database, the database can be protected.

6.2 Practicability

- On the GUI interface, each user can see their username, client id and the user which edits the whiteboard.
- On the whiteboard, each user is free to use the following operations: create a new canvas, write with a pencil, draw a line, draw a circle, draw an ellipse, draw a rectangle, erase the eraser, text box, switch colors, and switch width.
- For the manager, they have the freedom to control whether users are connected. If the user wants to connect, the manager needs to agree. If the user is already offline, the administrator can choose to disconnect the user.

6.3 Reliability

- The server has a log file that records all historical painting operations. Therefore, the manager can better manage the system. Even if the system crashes unexpectedly, the manager also can recover the system and fix the errors based on the log files.
- The server has the ability to control the client, such as refusing the client connection and disconnecting the client from the server, thus, the system can avoid illegal attacks by the client.

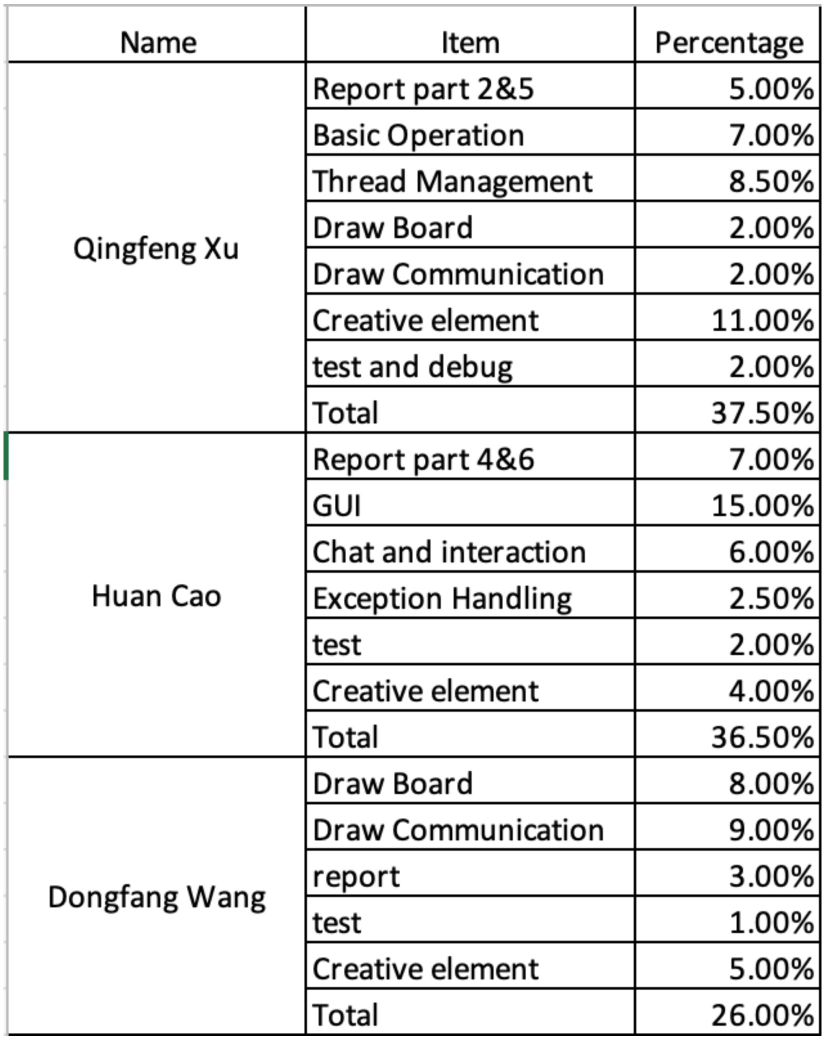
6.4 Efficiency

- This system transfers data by JSON to improve search speed. Also, The data format in JSON is simple and easy to read and write.
- The server has the function of controlling the client, which can reasonably disconnect the offline client. This function is efficient, and it is beneficial to the release of resources.

7 Conclusion

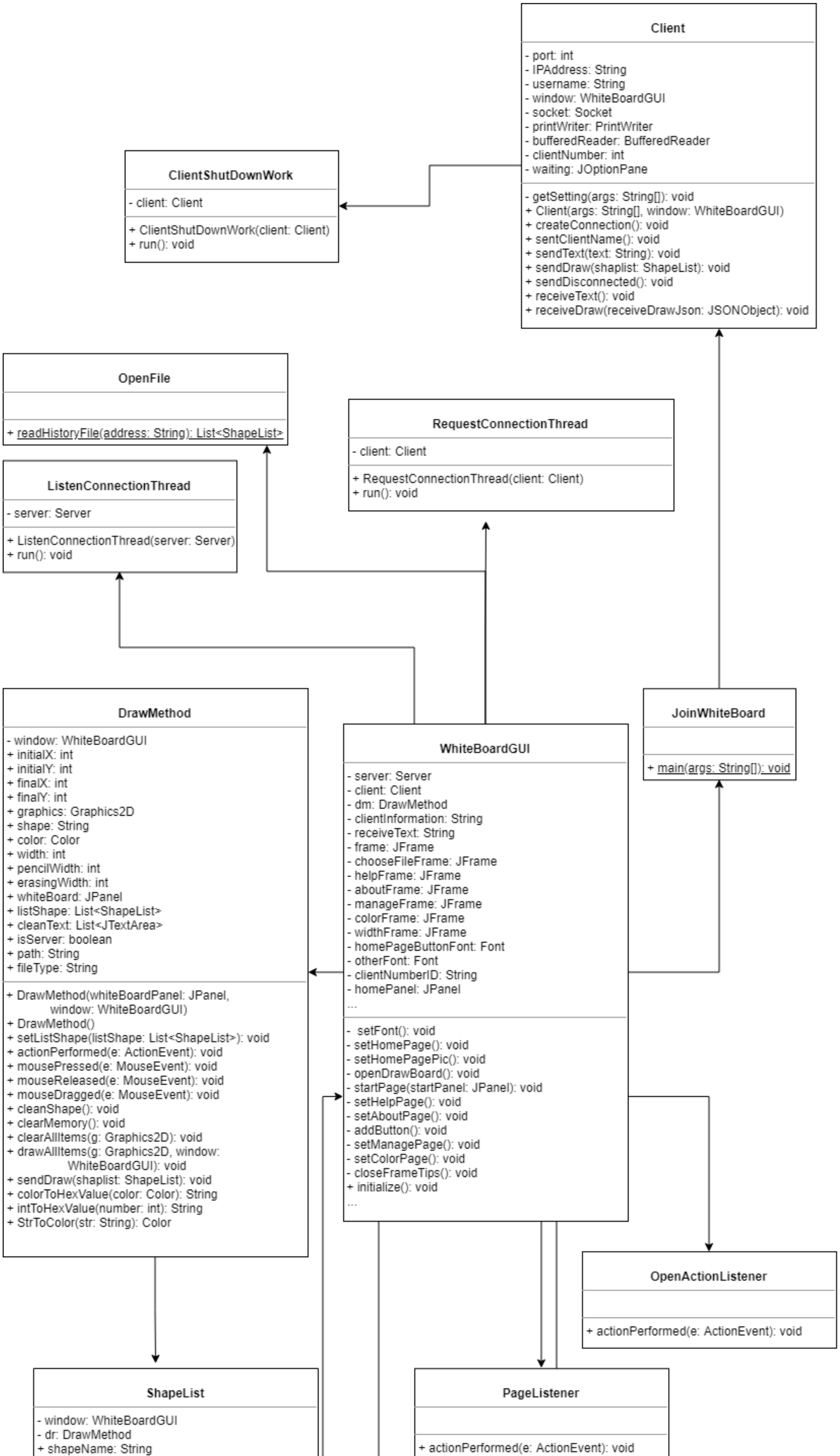
The system implements the shared whiteboard. The server provides threads for each connection and it connects to the client over TCP. Users can draw freely on the shared whiteboard and chat on the interface. The manager is free to control the user's connection. Also, he can create new files, open files, save files, save as files and close files. Concurrency processing, error handling, and system crash handling improve the consistency of the system; log files improve the reliability of the system. Moreover, JSON and thread management improve the efficiency of the system.

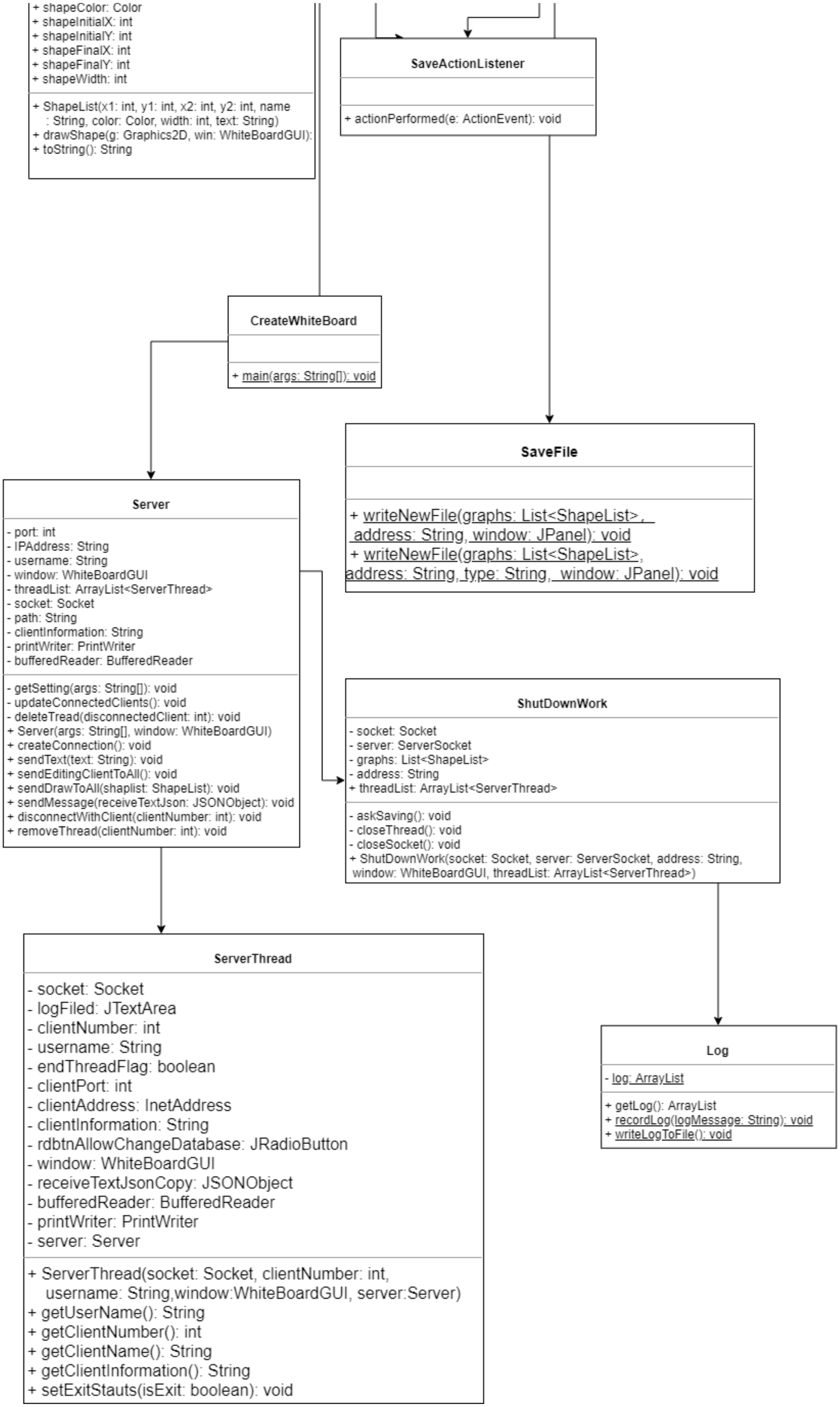
Appendix



Name	Item	Percentage
Qingfeng Xu	Report part 2&5	5.00%
	Basic Operation	7.00%
	Thread Management	8.50%
	Draw Board	2.00%
	Draw Communication	2.00%
	Creative element	11.00%
	test and debug	2.00%
	Total	37.50%
Huan Cao	Report part 4&6	7.00%
	GUI	15.00%
	Chat and interaction	6.00%
	Exception Handling	2.50%
	test	2.00%
	Creative element	4.00%
	Total	36.50%
Dongfang Wang	Draw Board	8.00%
	Draw Communication	9.00%
	report	3.00%
	test	1.00%
	Creative element	5.00%
	Total	26.00%

Task Break Down


Client
- port: int
- IPAddress: String
- username: String
- window: WhiteBoardGUI
- socket: Socket
- printWriter: PrintWriter
- bufferedReader: BufferedReader
- clientNumber: int
- waiting: JOptionPane
- getSetting(args: String[]): void
+ Client(args: String[], window: WhiteBoardGUI)
+ createConnection(): void
+ sentClientName(): void
+ sendText(text: String): void
+ sendDraw(shaplist: ShapeList): void
+ sendDisconnected(): void
+ receiveText(): void
+ receiveDraw(receiveDrawJson: JSONObject): void
ClientShutDownWork
- client: Client
+ ClientShutDownWork(client: Client)
+ run(): void
OpenFile
+ readHistoryFile(address: String): List<ShapeList>
RequestConnectionThread
- client: Client
+ RequestConnectionThread(client: Client)
+ run(): void
ListenConnectionThread
- server: Server
+ ListenConnectionThread(server: Server)
+ run(): void
DrawMethod
- window: WhiteBoardGUI
+ initialX: int
+ initialY: int
+ finalX: int
+ finalY: int
+ graphics: Graphics2D
+ shape: String
+ color: Color
+ width: int
+ pencilWidth: int
+ erasingWidth: int
+ whiteBoard: JPanel
+ listShape: List<ShapeList>
+ cleanText: List<JTextArea>
+ isServer: boolean
+ path: String
+ fileType: String
+ DrawMethod(whiteBoardPanel: JPanel, window: WhiteBoardGUI)
+ DrawMethod()
+ setListShape(listShape: List<ShapeList>): void
+ actionPerformed(e: ActionEvent): void
+ mousePressed(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+ mouseDragged(e: MouseEvent): void
+ cleanShape(): void
+ clearMemory(): void
+ clearAllItems(g: Graphics2D): void
+ drawAllItems(g: Graphics2D, window: WhiteBoardGUI): void
+ sendDraw(shaplist: ShapeList): void
+ colorToHexValue(color: Color): String
+ intToHexValue(number: int): String
+ StrToColor(str: String): Color
WhiteBoardGUI
- server: Server
- client: Client
- dm: DrawMethod
- clientInformation: String
- receiveText: String
- frame: JFrame
- chooseFileFrame: JFrame
- helpFrame: JFrame
- aboutFrame: JFrame
- manageFrame: JFrame
- colorFrame: JFrame
- widthFrame: JFrame
- homePageButtonFont: Font
- otherFont: Font
- clientNumberID: String
- homePanel: JPanel
...
- setFont(): void
- setHomePage(): void
- setHomePagePic(): void
- openDrawBoard(): void
- startPage(startPanel: JPanel): void
- setHelpPage(): void
- setAboutPage(): void
- addButton(): void
- setManagePage(): void
- setColorPage(): void
- closeFrameTips(): void
+ initialize(): void
...
JoinWhiteBoard
+ main(args: String[]): void
OpenActionListener
+ actionPerformed(e: ActionEvent): void
ShapeList
- window: WhiteBoardGUI
- dr: DrawMethod
+ shapeName: String
PageListener
+ actionPerformed(e: ActionEvent): void



+ shapeColor: Color
+ shapeInitialX: int
+ shapeInitialY: int
+ shapeFinalX: int
+ shapeFinalY: int
+ shapeWidth: int

+ ShapeList(x1: int, y1: int, x2: int, y2: int, name : String, color: Color, width: int, text: String)
+ drawShape(g: Graphics2D, win: WhiteBoardGUI):
+ toString(): String

SaveActionListener

+ actionPerformed(e: ActionEvent): void

CreateWhiteBoard

+ main(args: String[]): void

SaveFile

+ writeNewFile(graphs: List<ShapeList>, address: String, window: JPanel): void
+ writeNewFile(graphs: List<ShapeList>, address: String, type: String, window: JPanel): void

Server

- port: int
- IPAddress: String
- username: String
- window: WhiteBoardGUI
- threadList: ArrayList<ServerThread>
- socket: Socket
- path: String
- clientInformation: String
- printWriter: PrintWriter
- bufferedReader: BufferedReader

- getSetting(args: String[]): void
- updateConnectedClients(): void
- deleteTread(disconnectedClient: int): void
+ Server(args: String[], window: WhiteBoardGUI)
+ createConnection(): void
+ sendText(text: String): void
+ sendEditingClientToAll(): void
+ sendDrawToAll(shaplist: ShapeList): void
+ sendMessage(receiveTextJson: JSONObject): void
+ disconnectWithClient(clientNumber: int): void
+ removeThread(clientNumber: int): void

ShutDownWork

- socket: Socket
- server: ServerSocket
- graphs: List<ShapeList>
- address: String
+ threadList: ArrayList<ServerThread>

- askSaving(): void
- closeThread(): void
- closeSocket(): void
+ ShutDownWork(socket: Socket, server: ServerSocket, address: String, window: WhiteBoardGUI, threadList: ArrayList<ServerThread>)

ServerThread

- socket: Socket
- logFiled: JTextArea
- clientNumber: int
- username: String
- endThreadFlag: boolean
- clientPort: int
- clientAddress: InetAddress
- clientInformation: String
- rdbtnAllowChangeDatabase: JRadioButton
- window: WhiteBoardGUI
- receiveTextJsonCopy: JSONObject
- bufferedReader: BufferedReader
- printWriter: PrintWriter
- server: Server

+ ServerThread(socket: Socket, clientNumber: int, username: String,window:WhiteBoardGUI, server:Server)
+ getUserName(): String
+ getClientNumber(): int
+ getClientName(): String
+ getClientInformation(): String
+ setExitStauts(isExit: boolean): void

Log

- log: ArrayList

+ getLog(): ArrayList
+ recordLog(logMessage: String): void
+ writeLogToFile(): void