Shared white board

Introduction

This project is a shared white board program that allows multiple users to draw simultaneously on a canvas supporting drawing line, oval, rectangle, circle, free hand, texting as well as setting, chatting color and host can have some other operations like kicking the joined user and save or create current whiteboard.

Architecture Design

-Network Technique

Based on the requirements above, it has several ways to implement like using socket or RMI. RMI is based on Java's remote invocation mechanism, which hides the underlying network communication details. That allows method calls on Java objects to be executed on remote servers, making remote objects appear as if they were local objects. However, RMI is unnecessary for this assignment because the assignment is run locally. In addition, you need to design your own remote interfaces and servants, which is a bit complicated. For this assignment, the reason I use socket with TCP is that it is easy to implement receiving and sending messages and you can design your own exchange protocol freely based on the functions asked to implement.

-System Architecture

I decided to use client-server architecture model instead of using the peer process model. This is because in peer process model, there is no clear boundary that who is client or server. All participating nodes in the network are referred to as peers, and they can act both as clients and as servers. In this scenario, when each user does some operations, each of them would act like a server and send messages to each other. When more users join, it would be a bit messy to maintain connections and exchange information with multiple other peers. In addition, the requirement is to ask to have a manager role. As a result, I choose client-server architecture model and set the manager to be the server side as single central one for multiple users. You can see there are two packages guest for client, host for server in the source code files. The whole process happens based on the broadcast from the server side. That is the server (host) broadcast all operations to the client (guest) so that the client can simultaneously see all things on the host's whiteboard. In addition, when client does the operation, it sends message to the host, then the host replicates client's operation and broadcast this operation so that all clients can see this operation again.

-Exchange Protocols

In terms of the exchange protocol, I initially took JSON into account, after I searching several resources, I did not choose it because it needs to download some dependencies and it needs time to learn. For this assignment, using string is enough since it is easier to set message format, although it is a bit confused when use split () method to process the request. Data storage is also important, in this case, I create an array list to store the records of operations, and each record contains RGB color, start and end coordinates of x-axis and y-axis, drawing type as well as text content if texting on the white board. When the server (host) wants to broadcast the operations, it firstly need access this list.

Class Design and sequence diagram

- Server side

In server side, there are 10 classes in host package which are **Connection**, **ConnectionHost**, **Create, Host, Listener, OpenFile**, **SaveFile**, **SaveAsFile**, **Server**, **Painter**.

OpenFile, **SaveFile** and **SaveAsFile** class are served for File menu part. Because I design sperate UI for each option in the menu, they have their own classes to define the UI interface as well as some methods in their classes. I design to allow open selection in the menu to create **OpenFile** class for open file functionality and allow the whiteboard to open .txt file with valid records of operations. **SaveFile** class is used to save file and only save as .txt file to user's desktop. **SaveAsFile** class is similar but save additional file with selected file type and in this case, I save as a picture format like .jpg and .png. The reason I save two files in save as functionality is the preparation for open file functionality because open a picture is pretty complicated in the case of multiple users and cannot ensure there is no problem like showing the completed picture at the same time. Therefore, I save .txt file with valid record of operation and repaint this operation based on record.

Create is a runnable class and an entry to launch the **Server** class and initialize the whole UI of the whiteboard as a host, then **Host** class works to show the user interface. In **Host** class, it consists of three parts of UI, each of the parts should have the event-mechanism. Therefore, **Listener** class should be added to manage the operations from the whiteboard and give corresponding action to each operation and broadcast to all guests. **Painter** class is similar to Listener class, but only repaint based on the record list. The easiest way to manage event is to directly add action listener like addActionListener () method for each operation in the **Host** class. However, I only want the **Host** class to mainly have code part for UI. Therefore, by adding a new class **Listener**, it can manage and listen the action in a more unified module and is more convenient to maintain and modify in the future.

In **ConnectionHost** class, there are several methods, which the **Host** class can be used to update user list on the UI, check whether the requested joining user has the same name controlling the join flow, repaint and broadcast operations from guests.

The **Connection** is a working class that receive and process the request from the guests. Therefore, it should be designed to be the extended class for Thread because it is a server allowing for multiple connections from the guests which means that each guest has one connection. It can process a series of requests like joining, kick out, ask Host to repaint after do operations on its own whiteboard, texting on chat area and leave.

-Client side

In client side, there are 5 classes in guest package which are **Connection**, **Join, Guest, Listener, Painter**.

Join and **Guest** class are pretty similar to Create and Host class. There are several main differences. First, there are no kick functionality and file menu in the **Guest** class which is the UI part for guest

whiteboard because guest has no access to do these operations, so I delete them. Second, because I design a pop-up window that host can choose to allow joining or not, in the Join class, after click join button, it receives the permission from the host and determine whether to initialize the guest UI interface.

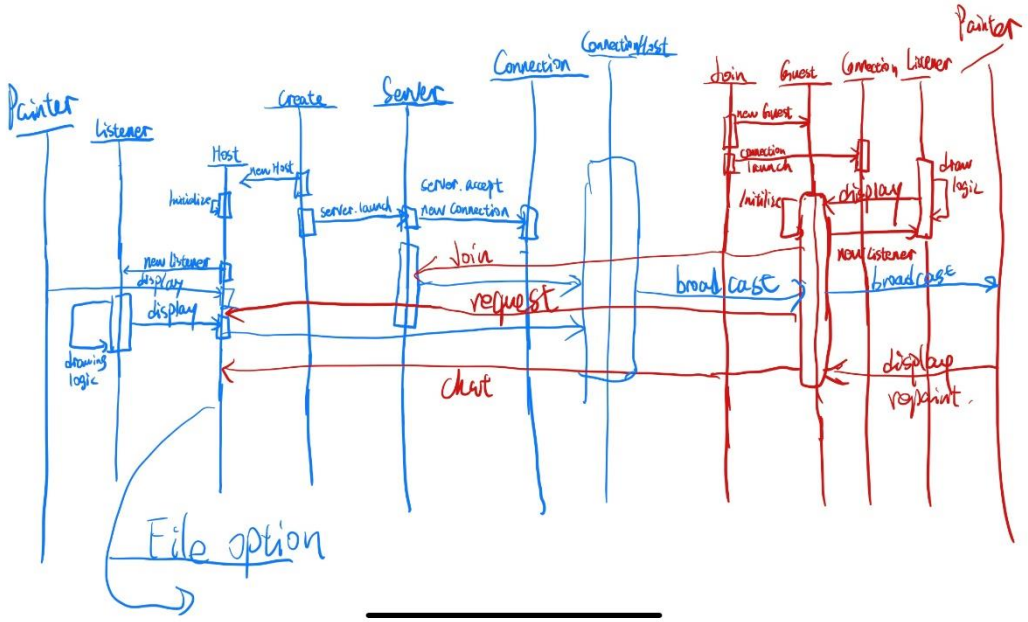
Painter class is the same as **Painter** class in server side. However, the Listener class has one extra function sendDraw(). This is because I choose to use broadcast design. So, after each operation, the guest should tell host and host is responsible to broadcast this operation to other guest.

-Concurrent issue

Concurrent issue is very important in this assignment since it is a shared whiteboard that multiple users can do operations at the same time. The server should be capable to handle this. In **ConnectionHost** class there are several synchronized method which are **addUser()**, **checkIn()**, **canvasRepaint()**. **addUser()** method is to add the user to the UI interface and this method should be synchronized because it can avoid errors when there are too many users joined. So does the **checkIn()** method which is to check whether the requested joining user already has the same name in the user list. The reason I did not consider the user deletion to be synchronized is because the user can leave at any time so no need to wait. **canvasRepaint()** method is also to be synchronized because it would call update() method in the Listener (host) class that adds operation to the record list and call repaint() method in Painter(host) class, the operation from the user. If too many users do operations on their whiteboards and ask host to repaint and broadcast their operations again at the same time, errors may happen.

Sequence diagram

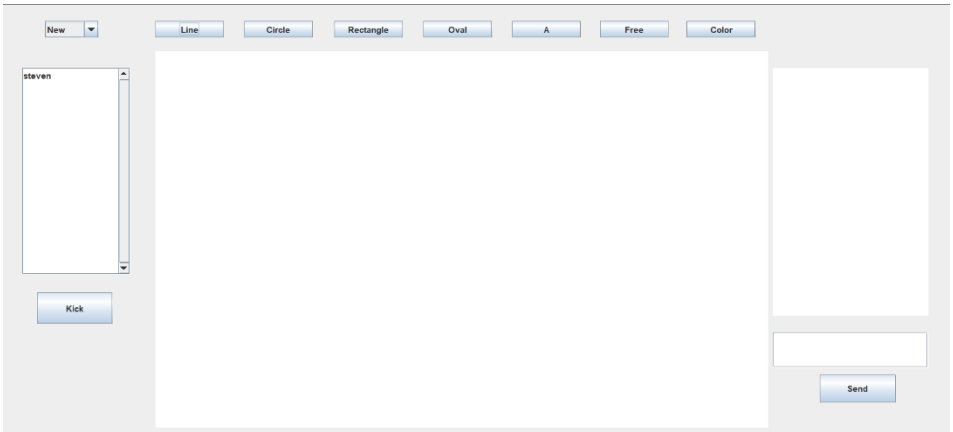
Below is the flow chart of the whole project. The **Create** class launch server and ui, then the server listen the guest connection and their requests. Host and guest all have listener as event-mechanism and painter to repaint operations from each other. ConnectionHost is used to broadcast message and things to all guests.



UI Design

I design the UI interface for the whole white board including the main body of the white board with several functional buttons, kicking, chat area as well as the menu for saving part.

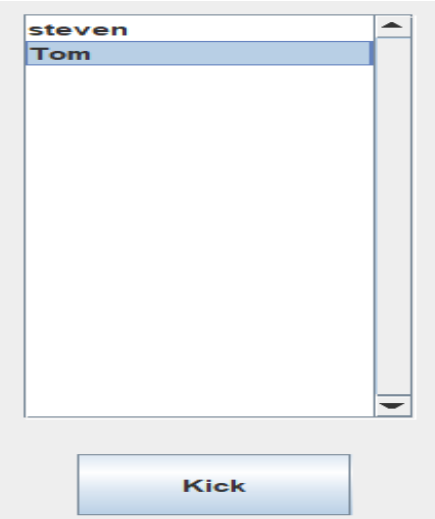
-Main body



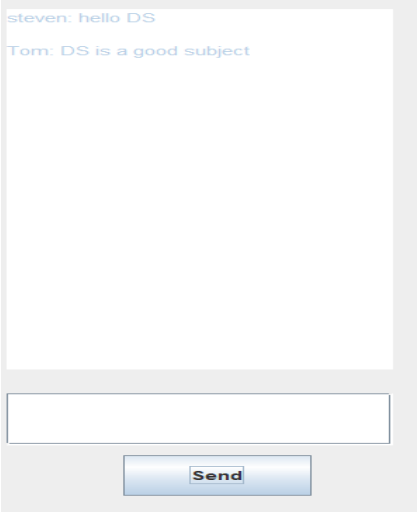
The interface is the whole UI lay out of the assignment. You can see there are several buttons on the top of the whiteboard which are Line, Circle, Rectangle, Oval, A, Free and Color. The first 4 buttons can be used to draw corresponding shapes on the whiteboard. Button A is to implement texting on the board. After click this button, a texting area would pop up and you can text whatever you like on this white board. Button Free is used to freely draw patterns on the whiteboard when the mouse is dragged on the whiteboard. Button Color allows user to choose their favorite color to draw the above features. After click it, a color panel would pop up and you can choose the color you want.

-Kicking

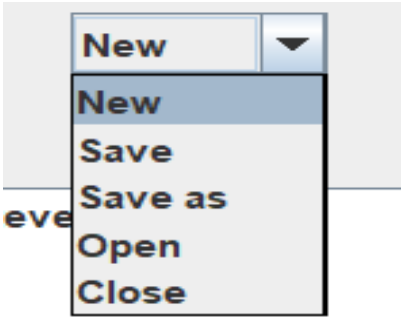
This is the kicking part which consists of two parts. The first is to display the list of users currently joining the whiteboard. By combing the JList with JScrollPane in the java.swing package it is easy to implement this functionality. The second is to kick out the user based on the selected user in the JScrollPane after pressing the kick button. Then, a window would pop up to tell all users who has been kicked. Even the user who has been kicked out would also receive a message to tell he/she has been kicked out by host.



-Chat area



This is the interface that allows user to have communication with each other. This consists of two parts. The first is to show the chatting content, you can see a chat box showing who sent the message. The second is the text field and send button, which users can text the message they want and send to the chat area that everyone can see it after clicking the send button.



-Menu

This is the menu part. As you can see you can find several options New, Save, Save as, Open, Close in this menu. I design this menu by using JComboBox in java.swing since it looks like the real design in our true life.

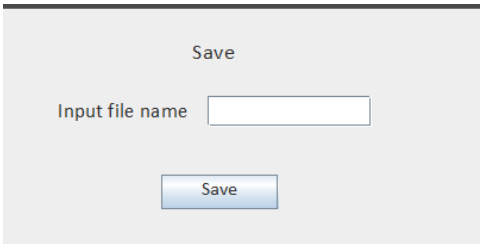
New selection here is to clear the whiteboard to looks like it creates a new whiteboard. Close selection will close thehost's whiteboard and all users quit.

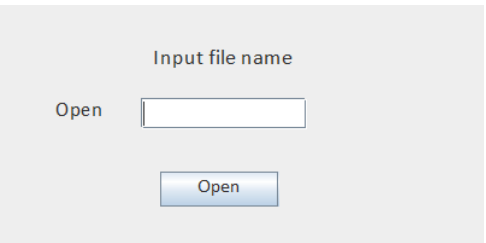
Open selection is to open the .txt file which includes valid records of operations for repainting. Based on the records, it can repaint the previous saved work on the whiteboard.

Save selection can save the valid records of operations to the user's desktop locally. After click it, a window would pop up that you can text file name you want to save and it can only save file as .txt format to the user's desktop.

Save as selection can save the content on the whiteboard to be different format. However, this time I design to save as .jpg and .png format and it can extend data format in the future. The picture of the content on the whiteboard would be saved to the user's desktop after choosing .jpg or .png. In addition, it also saves another .txt file for the use of Open selection.

These are user interfaces when select Save option, Save as option, Open option.






Save As
Input file name
.jpg
.jpg
.png
Save