Design

Datatype

GUI DESCRIPTION:

Our GUI will consist of two components: the "Control Panel" and the "Workspace." Upon connecting, a user will be asked to put in a username. The Control Panel will then appear and have the list of active whiteboards and the users currently connected. From here, a client will be able to join a whiteboard or create a new whiteboard. This will spawn a new Workspace window, where the whiteboard and drawing tools will be available. The client will only be able to have one whiteboard open at a time.

CLIENT-SERVER DESCRIPTION:

Each client will have one whiteboard locally, which is pulled from the server. The server will maintain the main copy of each whiteboard. Whenever a client performs an action, the action is shown on the the client's local whiteboard immediately. The action is also put onto an outgoing message queue of requests to be sent to the server. When the server responds with an updated image, it will be put onto the client's incoming queue of image updates. The client's whiteboard will refresh from this incoming queue periodically and will be drawing on top of the latest one in the queue.

When multiple clients modify at the same time (i.e. one draws where the other erases) the server's whiteboard will not know, because we will put all incoming messages onto a single event handling queue. Therefore, the edit which reaches the server later (and is put later in the queue) may override another regardless of the order in which the clients sent them.

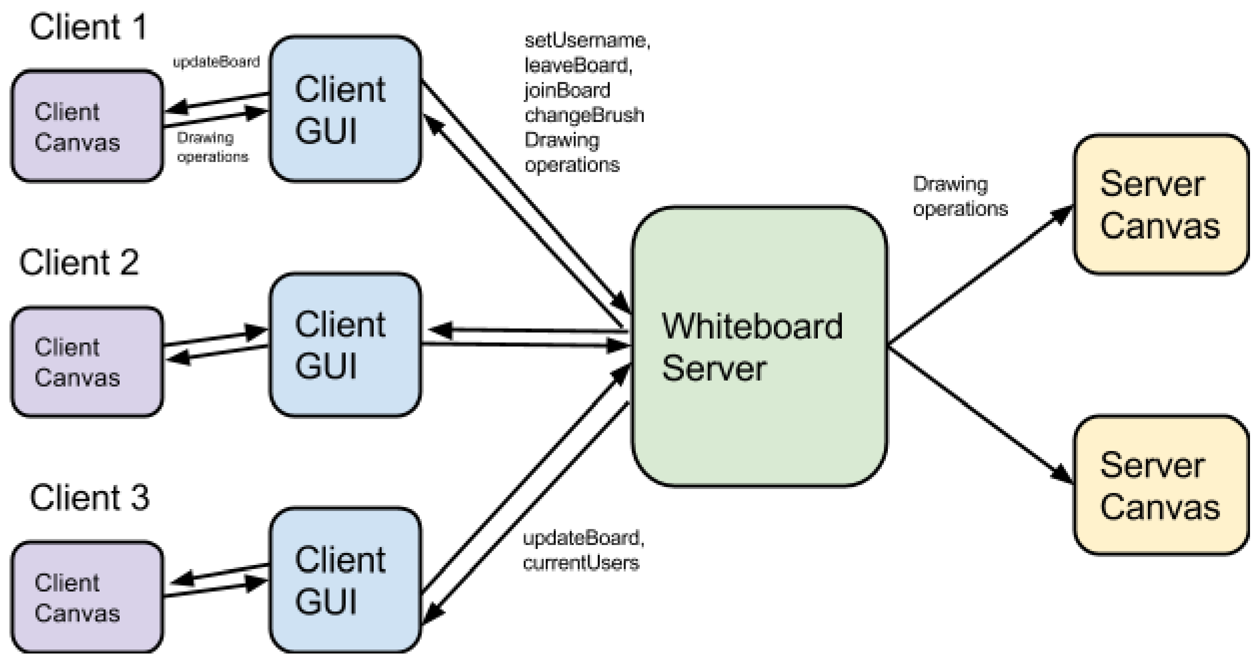
IMPLEMENTATION:

ClientGUI: A GUI class that contains the information of the available whiteboard to edit, as well as the users connected to each whiteboard. This class will initialize and maintain the connection with a WhiteboardServer

ClientCanvas: A class that allows the client to draw on a whiteboard using different tools and methods. Messages will be generated and sent on every mouse release.

WhiteboardServer: A server class that handles all of the incoming client to server messages, as well as managing and updating all of the collaborative whiteboards and connected users. Contains a ServerCanvas to represent each whiteboard. Each connected client will have a single thread where their current information is stored (what brush size/color, what whiteboard they are editing). Each of these threads will put any incoming messages onto the thread of a ServerCanvas to make edits to a whiteboard.

ServerCanvas: A class that will represent the server's centralized model of a single whiteboard. All changes to this whiteboard will be done through a single thread, to prevent concurrency issues



Protocol

Using telnet, clients connect to a centralized server and send commands. Each request will have its own flags and values. Here are a few central requests (Client to Server):

```
setUsername [username]
listBoards
joinBoard [boardId] //this command will create a new board with this ID should it not
exist
changeBrush -c [color] -w [strokeWidth] -f [true|false]...
leaveBoard [boardId]
```

Drawing Operations:

```
[operation] (points)
drawLineSegment x1 y1 x2 y2
eraseLineSegment x1 y1 x2 y2
drawRectangle x1 y1 xLen yLen
drawOval x1 y1 xLen yLen
eraseAll
```

The server responds to every drawing operation by sending ALL connected clients a String representing the last drawing operation. The Strings of operations will be placed in a queue to ensure proper order (such as overlapping). A separate method will analyze the Strings and rebuffer the whiteboard accordingly. Here are a few responses (Server to Client):

```
currentBoards [boardID1][connectedUser1,connectedUser2,...]
[boardID2][connectedUser1,...]...
updateBoard [ArrayList] //each client will only ever be connected to one whiteboard, so
will                                only receive this message for their connected board
```

Concurrency

Each of the clients will be connected to the server on their own thread. They will update information about their current brush state (color, width, etc.) and the server will store this information in the thread connection with the client. This information is thread safe because only that client will ever be updating this information. The client will also pass messages for edits to make to the whiteboard. The server will have one event handling thread for each of it's own whiteboards, forcing all edits to each whiteboard to be single-threaded. The UI won't be guaranteed to reflect all changes from other clients at all time.

Testing Strategy

The ClientCanvas and GUI have user interface components that will be tested manually. These classes will also have methods that send/receive messages to the server, so we will separate these message-sending methods so that we can write automated tests for these.

Areas to test:
-Can join and leave boards multiple times
-All drawing operations function properly
-Usernames update when other clients join/leave
-Each outgoing message is properly formatted
-Receives and parses messages properly

We will manually test WhiteboardServer and ServerCanvas for concurrency issues by simultaneously connecting with multiple users. These classes do not have user interface components that will need to be tested We will also write automated tests for any message-generating method, and make sure that the server can parse messages properly as well

Areas to test:
-Multiple clients can connect to same whiteboard
-Multiple clients can connect to different whiteboards
-Each outgoing message is properly formatted
-Receives and parses messages properly
-Organizes incoming commands into a single thread
-Organizes outgoing commands properly based on the commands that have come in

(Defining whiteboards. One of your first tasks is to determine what whiteboard is in your system. Is a whiteboard more like a 2D array of pixels, or more like an arrangement of shapes? How are they named? What drawing operations are available, and what does it mean to remove or erase a drawing? What happens when two users draw on or modify the whiteboard simultaneously? Will both of their edits be reflected or might one override the other? Are some changes independent and some interfering, and if so, how is interference resolved? The most important piece of advice here is to **keep it simple**. You can always extend your program, but if you've created a complex mess, it's hard to go back.)

(Designing architecture and protocol. You must also devise a network architecture and a protocol for this project. A client/server architecture is potentially the easiest choice here, but it isn't required. You should strongly consider using a text-based protocol, which is generally easier for testing and debugging. We used text-based protocols in the problem sets. Services that use plain text protocols — e.g. HTTP or SMTP — can talk to a human just as well as another machine by using a client program that sends and receives characters. You've already used telnet as a tool for interacting with text-based protocols in the problem set. Make use of it here too.)

(Design for safe concurrency. In general, making an argument that an implementation is free of concurrency bugs (like race conditions and deadlocks) is veryt difficult and error-prone. The best strategy therefore is to design your program to allow a very simple argument, by limiting your use of concurrency and especially avoiding shared state wherever possible. For example, one approach is to use concurrency only for reading sockets, and to make the rest of the design single-threaded. And recall that even though user interfaces are concurrent by nature, Swing is not thread-safe. Recommended reading: Threads and Swing.)

Handling multiple users. Since realtime collaborative whiteboard is not realtime without at least two people, your system must be able to handle multiple users connected at the same time. One reasonable design approach could be using one thread for reading input from each client and having a central state machine representing the state of the server (using one more thread, to which each of the client threads pass messages through a shared queue).