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.

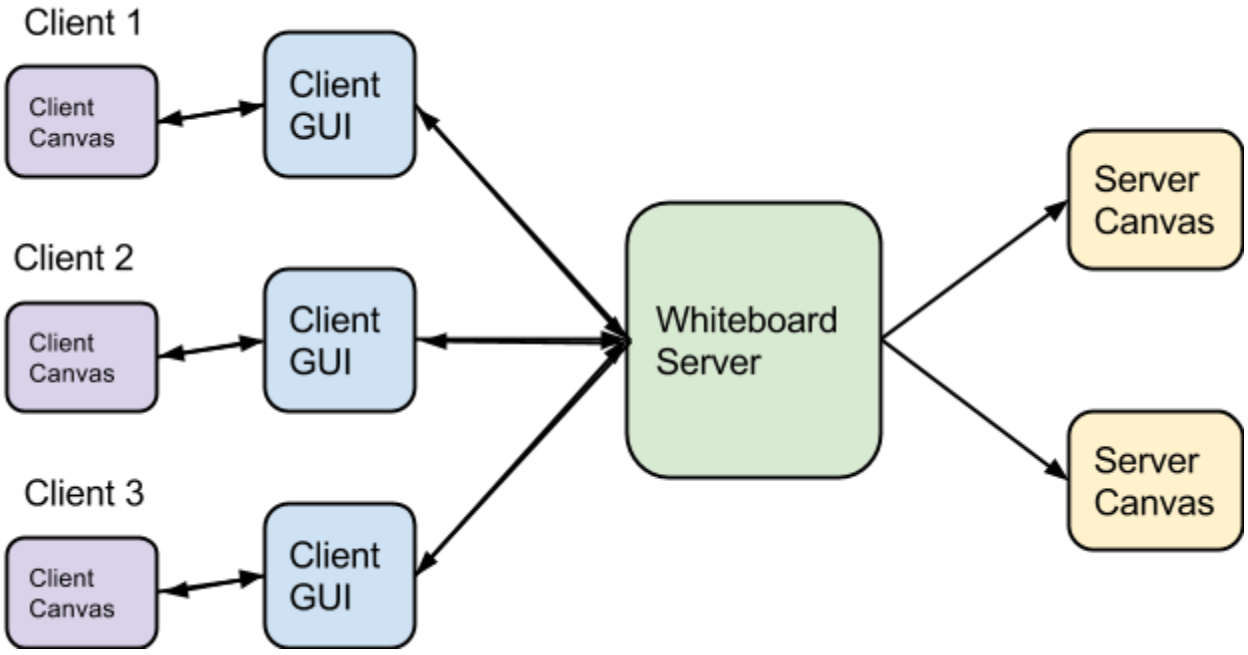
IMPLEMENTATION:

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 byte[][] representing the resulting image. Here are a few responses (Server to Client):

```
currentBoards [boardID1][connectedUser1,connectedUser2,...]
[boardID2][connectedUser1,...]...
updateBoard [byte[][]] //each client will only ever be connected to one whiteboard, so
```

will only receive this message for their connected board

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).

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

Design for testability. To make it possible to write unit tests without having to open socket connections and parse streams of responses, you should design your components so that they can be driven directly by a test driver — e.g. by calling methods or by putting messages into a queue.
Testing GUIs is particularly challenging. Follow good design practice and separate as much functionality as possible into modules you can test using automated mechanisms. You should maximize the amount of your system you can test with complete independence from any GUI.

The JUnit tests that we will create will focus on message driven actions. For example, we will test the message-passing between the client and the server by calling the specified method directly. This way, drawing actions could be tested as well. Several modules could be tested automatically, such as checking whether concurrent actions by different clients correctly override each other in the same whiteboard.