What we need in order:

Specifications:

Data Structure: Client, server, Brushstrokes.

Protocol: How server and client will interact.

So in the boards, we’ll need to either use a dictionary to store boards and their associated strokes, or actually set up a field in the board class itself to hold all its associated strokes. Every board will also need to maintain a list of users associated with it. We should create a Board class which contains an ArrayList of strokes, a list of users, and a canvas.

Users will have a Canvas, the GUI. The data representation of the user uses the list of strokes, listens for a server announcement of a new stroke on the board. Right now, we have a canvas, which maintains both the state of the image and a view, which updates itself based on the view we obtain.

There is currently no list of strokes in the user client side, so that needs to be implemented.

We currently have a listener on the canvas so that when the user drags across the screen, we create a new Stroke object and send it.

We can use a Queue to support the canvas instuctions, and send Brushstrokes across the network. We’ll have one thread for the queue and EDT, one thread for the handling of messages, and one thread per board instance.

Server side, we would have one thread per board, and one thread to continuously monitor the queue/input strings and keep everything updated. The input from this thread will be made into a Stroke object, if not already one, and added to a blocking synchronized array list. We’ll most likely need to use JavaListener to listen for change

Make all the input streams threads to put them onto the Queue. The info from the queue is used to create Strokes which are added to a blocking synchronized array. The server’s threads can all pull things from the array, and then update the boards with these Brushstrokes.

We need both a Board ID to ensure message delivery to the proper board, but we don’t need user ID as we know which board the message came from, and that’s the only important thing. We don’t need to know who drew, just that someone drew at a specific time on a specific board.

If we decide to make multiple boards available for one user, then we’ll have one thread per socket per output stream, and we’ll need a queue on the output stream to support that.

Make boards independent of protocol or threading like in PS3, all the threads from the different clients become the controllers for the board, and the board itself is our model. We don’t really have a view on the server, but we do need to test that model and communication between server and client.

On the client side, the GUI is the view, the state of the image as in strokes is the model, and the listeners for input from the server are the clients.

We also need a join method, and a decision:

If we want, we can:

1. Initialize either a list of boards at start and request a user to declare which board they want to connect to at the start and then require reconnections in order to get them to switch boards and implement a board switcher in the toolbar.
2. Initialize new boards whenever requested and also request a user to declare the board they want to connect to or if they wish to create a new board.

Regardless of the above, we’ll have to keep track of boards and make sure that if a user requests a non-existant board, we catch the exception that’ll throw and report that that board cannot be selected.