NOTE: Many of the commits were made from krwang's computer but the work was actually more balanced than the commits indicate. Since a large portion of our debugging and implementation (especially with message passing) involved the client and server code, we decided it would be easiest to work from one computer; this prevented merging issues and made sure nothing was messed up when pushing. However, we did not specify who did each commit from krwang's computer but we definitely did split the work evenly.

Major Changes To Design:

- Instead of sending the whole canvas image when connecting to a canvas, the server now sends a stream of drawing actions that have been performed on the canvas. The client then opens a new canvas and draws the actions from the server's output stream.
- We separated the requests clients can send into distinct methods (addRequest(), drawRequest(), byeRequest()) instead of a single sendRequest() method.
- In various classes (most importantly WhiteboardClient), we added fields necessary for message passing and threading
- Adjustments to the syntax for message passing to include necessary information
- Added a UsernamePanel class that is incorporated into the WhiteboardGUI. This panel contains the list of users currently accessing the canvas through the server.
- Modified EntryGUI to be simpler and more streamlined
- Allowed one user to access multiple canvases at the same time

Datatype Design: We will be using a client server model

Client Side: We will be using a Model View Controller pattern. The Model will be the Canvas, the View is the whiteboardGUI, and the Controller is the whiteboardClient

WhiteboardClient: (controller)

- Spec: this class will receive updates from the Canvas and pass them on to the Server. It will then receive changes from the Server that will be reflected in the Whiteboard GUI
    - *Fields:*
        - WhiteboardGUI gui: keeps a copy of the Whiteboard GUI so it has access to the Canvas and the current state of the tools panel
        - Socket socket: stores the socket that is created for connecting to the server
        - String canvasName: stores the name of the canvas the client is currently connected to
        - String username: for the server to identify which client is passing a message to it
        - BufferedReader dataIn: for reading outputs from the server
        - PrintWriter dataOut: for message passing to the server
        - String ipaddress: the IP address of the socket connection
- Communication to the Server (addRequest(), drawRequest(), byeRequest()): WhiteboardClient processes events from the GUI, including joining and exiting a canvas,

and sends them to the Server
    - The Canvas listens for actions from the user, and passes the actions to back to the Client for processing
    - The Client then updates its local image in the Canvas (via a draw() method), and sends this local update (using our brush action syntax) to the Server
    - The rest of the processing is described in the Server documentation
- Communication from the Server (handle() message): WhiteboardClient processes drawing events sent from the Server and updates local image. This provides real time updates on actions other clients perform on the canvas
    - The WhiteboardClient receives an action from the Server with the parameters of the action (x1, x2, y1, y2, brush size, brush color, canvas name)
    - The WhiteboardClient processes this action using the WhiteboardGUI's draw() method, which updates the Canvas image using the provided arguments
- Clarification: The WhiteboardClient that sends an action automatically updates its own snapshot locally, and then all other WhiteboardClients connected to the same Canvas receive an update message from the Server
    - This communication and processing will be threaded (anonymous threads)
    - The protocol used follows a loose publish-subscribe pattern between all clients connected to the same Canvas, where one WhiteboardClient publishes a message and all other connected WhiteboardClients are subscribers that receive the message after it is processed and sent out by the server

WhiteboardGUI:(view)

- whiteboard GUI's will contain the settings the user has currently selected (brush size, brush type, brush color).
    - all elements of the GUI are local, with the canvas updating according to server messages; tools panel and username panel cannot be affected by the server
- this will also contain a switch button: if the user clicks it, they will be able to choose a new whiteboard to open and then their current whiteboard will close
- Spec: this class combines the canvas, tools panel, and username panel in one jFrame to present a unified appearance for the GUI
    - it will also keep track of the current users and display their usernames. the users will be stored on the server, which will perform checks when users join to ensure no duplicate usernames on a given canvas
    - each client can only have one whiteboardGUI open at a time
    - each whiteboardGUI can only contain one unique canvas
    - *Fields:*
        - Canvas
        - Toolspanel
        - Usernamepanel
        - switchButton
- Communication between the GUI and the Canvas (View and Model): the whiteboard client will act as the controller and will pass updates from the GUI to the canvas and the server.

The client will also receive messages from the server and update the canvas, which is then reflected in the GUI

Canvas: does not deal with the gui (model)

- Purpose: this class represents the actual surface upon which the user can draw with their mouse. It also stores the image on the screen, which will be saved so that when the user reaccesses that specific instance of the canvas, the image will be saved with it
- The canvas is completely locally created. However, when accessing a canvas on a server, the server will pass all previously performed actions to the client, who will then process those actions and update the canvas to match what other clients accessing the canvas see
- Spec: The Canvas class is the representation of the client's drawing
    - *Fields:*
        - Image: this is the image of the entire drawing on the canvas with changes from multiple users
- Clarification: The Canvas contains both the image and the listeners, but the two never interact with each other directly, only through the WhiteboardClient as an intermediary
    - In this sense, the Canvas acts as a sort of model because it stores the Image data, but it also acts as a view in its processing of user actions through listeners

Concurrency: The user's own changes to the canvas will be reflected on the user's own whiteboardGUI first (locally), and then other users' changes will be reflected as the server passes messages back to the users connected to the canvas

Server Side: We will be using a producer consumer model to create a Queue that stores all of the actions performed by all clients on various whiteboards. In this case, the client is producer, and the server is consumer
Spec: creates a serversocket to a specific port and then listens for the clients that are connecting to the server

- *Fields:*
    - canvasMovesMap: hash map that stores the actions that have been performed on each canvas on the server.
        - Maps: String canvasName ===> ArrayList<String> moves
        - to create a new canvas, the client must load a canvas under a unique name; creating a canvas under a previously used name will automatically load the previously loaded canvas's moves
    - sockets: hash map that stores the sockets connected to each canvas on the server
        - Maps: String canvasName ===> ArrayList<Socket> sockets
    - usersOnCanvas: hash map that stores the usernames of clients connected to each canvas on the server
        - Maps: String canvasName ===> ArrayList<String> usernames
        - usernames cannot be repeated when accessing a canvas, that is, all

usernames accessing a single canvas must be unique.
- the server will reject duplicate usernames

  - queue: an ArrayBlockingQueue where messages from the clients are stored
  - thread: the thread for handling messages from clients
  - serverSocket: The serverSocket that clients can connect to
- the inputs received are one of three strings:
  - “username “ + boardname + username: a client attempting to open up a WhiteboardGUI from the EntryGUI.
    - if the username is already taken in usersOnCanvas.get(boardname), the server will send a rejection message to the client, who will see a JDialog indicating the username is already taken
    - if the username is not taken, an approval method is sent to the client and a WhiteboardGUI is then created
  - “add “ + boardname + username: a client attempting to connect to the server and create/load a canvas under boardname
    - if either boardname or username is empty, the input will be rejected locally
    - if username is not unique from all other usernames connected to boardname, the server will reject it
    - otherwise, the server will send all previously performed actions on the canvas under boardname and store the username in usersOnCanvas.get(boardname)
      - the server will also update the currently connected clients, sending the new username that has connected
  - “draw “ + x1 + y1 + x2 + y2 + brushSize + brushColor + boardname: a client drawing on a canvas from (x1, y1) to (x2, y2) with a brush of size brushSize and color brushColor (if the user is erasing, brushColor = Color.WHITE)
    - the server will add this request to canvasMovesMap.get(boardname) and send this move all users connected to the canvas under boardname
  - “get “ + boardname: a client is loading a canvas and requests all previous actions on that canvas to preload its image
  - “bye “ + boardname + username: a client attempting to disconnect from the canvas under boardname (this message is sent when the Whiteboard GUI is closed.
    - the server will remove username from usersOnCanvas.get(boardname) and send a message to all clients connected to the canvas under boardname that username has disconnected
    - the server will also send an “end” message to the exiting user to allow closing of sockets

contains the following methods:

- serve(): runs the server and listens for the connection of clients to the server and handles their inputs by referring to the handleConnection method
  - we will be creating unique threads for each socket so that each client has one unique socket and won’t overlap in data input and will thus be threadsafe

- handle(): reads client's input and puts it in a queue(the input will consists of the protocol, the socket name)
- handleRequest(): will handle run the methods that should occur for different protocol commands sent to it (open, bye, draw, erase)
- sendRequest(): sends the output of the handleRequest() back to the client

ToolsPanel:

- Purpose: Allows users to customize their current drawing settings (brush type, brush size, brush color)
- Part of the overall GUI
- All functions contained in action listeners, as the panel responds to user actions to customize their brush settings

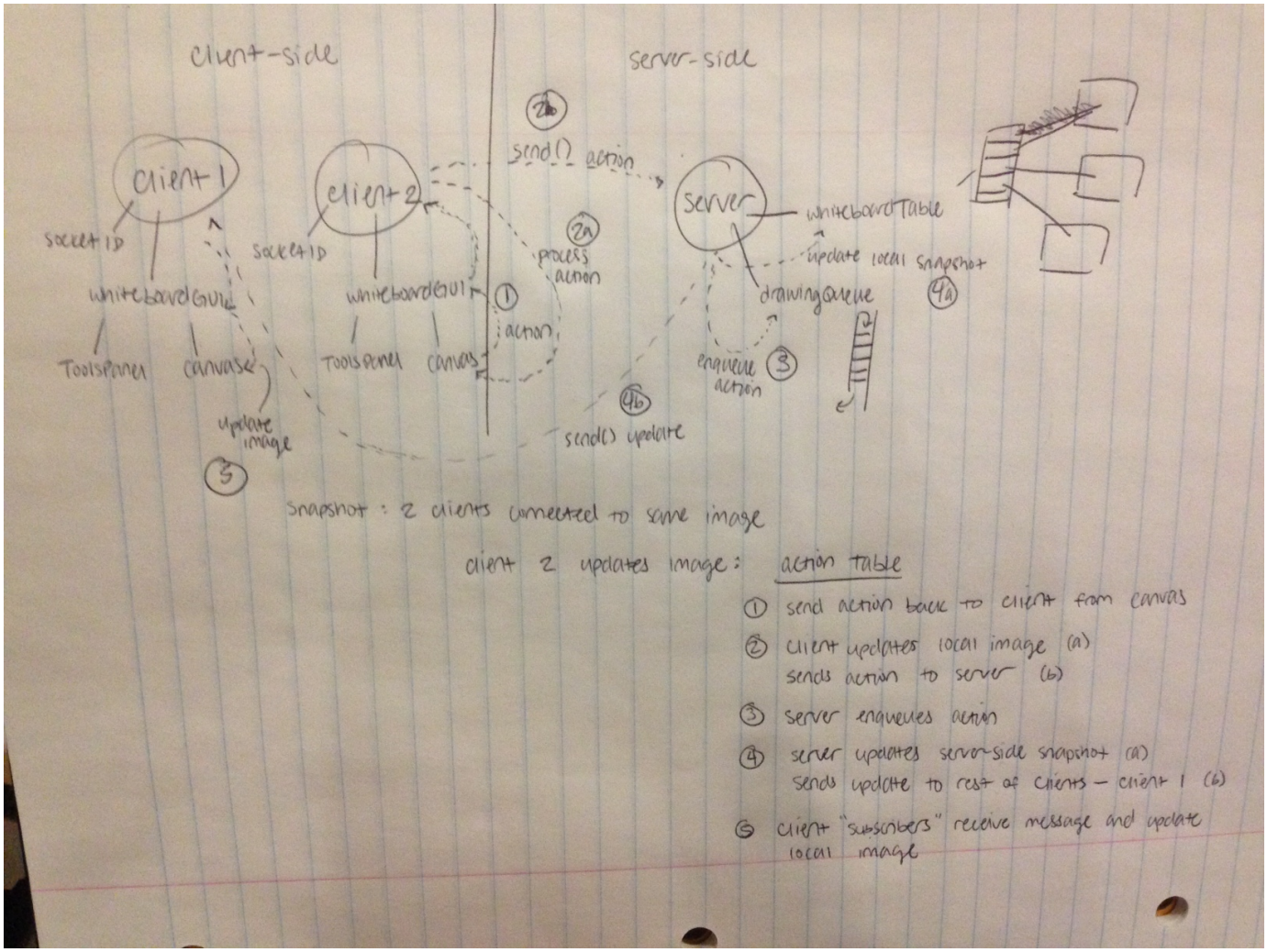
UsernamePanel:

- Purpose: Allows the users to see the other users currently connected to the canvas
- Part of the overall GUI
- Doesn't contain any listeners, as all updates will be sent from the server

EntryGUI:

- Purpose: GUI for users to specify whether they want to create a new Whiteboard on the server or load a previously created Whiteboard from the server
- When starting the EntryGUI, the user has the option of providing an IP address for remote connections. The default IP is localhost (127.0.01), but if the server is on a different machine, the user may enter that host's IP address instead.
- When creating a new Whiteboard, the GUI prompts the user to enter a name for the canvas that the server will store the canvas under. The name for the canvas must be unique to all other canvas names in the server
- Contains a definition and constructor for a NameGUI instance, which is the prompt for the user to input a name
- The user will also have to input a unique username that will be sent to the server as an indication of the number of people accessing each canvas
- public EntryGUI(): constructs a EntryGUI

Snapshot Diagram:



Protocol: designing how client and server interact

- the client will be passing information to the server by using a text based protocol, and we will be using telnet to test the protocol
- the client to server protocol *(see above in WhiteboardServer for server responses)*
    - “username boardName username”: sends request to server to check if the username is available for the canvas under boardName
    - “add boardName username”: sends request to server for the canvas with the specified name
    - “draw x1 x2 y1 y2 brushColor brushSize boardName” :this can only be written if the canvas has already been loaded on the client side
    - “erase color size x1 x2 y1 y2 whiteboardname”: this can only be written if the whiteboard has already been created.
    - “bye boardName username”: this can only be written if the x button on the

WhiteboardGUI is clicked
    - “get boardName” requests the list of previous actions on the canvas to be loaded, thereby preloading the last saved image
- Server to Client Protocol:
    - “draw x1 x2 y1 y2 brushColor brushSize boardName”
    - “add boardname username“ will send a message with a username. When necessary, the server will send this for all usernames connected to the canvas under boardname (reading from usersOnCanvas) and the client will update its usernamePanel to display all connected users
    - “bye boardname username” indicates to clients that username has disconnected from the canvas, so the usernamePanel is updated to reflect the disconnection
    - “end” is the sentinel value that exits the message handler and allows the client to close all streams and close the socket before exiting the program

Concurrency Strategy:

- We would like to have all users be able to draw on whatever whiteboard they choose concurrently
- There cannot be any deadlocks because users are not dependent on other users, there is no dependency between classes
- Another concurrency problem would occur if two users are trying to draw on the same whiteboard at the same time because the users could have different settings with which they’re drawing
    - we will solve this by determining the order of the actions by which one reaches the server action queue first.
- We will employ a client-server design implementation along with a confinement strategy to prevent multi-user access to anything but the shared Canvas object, which will in turn ensure thread-safe behavior for all datatypes aside from the Canvas class (the thread safety argument for the Canvas class is described above)
- We will employ a client-server design implementation along with a confinement strategy. Also, clients do not have direct access to anything on the server; they must send an action to the server to get appropriate information (users, moves on canvas) from the server.
- There are no elements that have to be safe for use by multiple users; all accesses and updates performed on a canvas are done through message passing between the server and the client
- Essentially, each whiteboard client will first update its on whiteboard GUI with the changes that its client made, and then the additional changes from other clients will be

made a little later in order in which they were received in the server queue.
    - This may affect the overlap of lines, depending on which change reaches the server queue first (a race condition that we are allowing)
- We will be updating the whiteboards instantaneously to prevent merge conflicts between the user's initially different versions of the same whiteboard

Testing Strategy

- **We will not be using JUnit tests in our project**
- **We will be testing the model, networking, and gui**
- **Model:** to test the model, we will be testing the Canvas class:
    - test that the strokes drawn by the user are accurate on the Canvas's image by comparing the location of the stroke with the location on the Canvas
    - test that the model is named correctly according to the user's input for the whiteboard name
    - test that the username list contains all users currently using that canvas
    - test that the correct image is drawn when the listeners are invoked for action events on the canvas
- networking: to test the networking, we will test several parts:
    - testing the protocol: we will test that all the commands result in the right text commands for the protocol
        - we will also test that all text commands result in the right action according to the protocol
    - testing that the server's canvas image is identical to that of the different clients that are all working on the same canvas whenever different clients draw on the same whiteboard
    - testing the sending of a draw message
    - testing the sending of an erase message
    - test the sending of a bye message
    - test the sending of an open message--if the whiteboard is already created, or hasn't been created yet
    - **Server:** Using JUnit tests we will test all messages that can be passed to the server and check that the server returns an appropriate response.
        - test opening/closing/loading a Whiteboard
        - test opening/closing/loading multiple Whiteboards (make sure the server keeps track of which edits are performed on which Whiteboards)
        - test one user editing a Whiteboard
        - test multiple users editing one Whiteboard (checking for race conditions?)

        - testing having up to 3 users editing one Whiteboard
        - test accessing a whiteboard that has already been created in the past
    - test when opening and closing same whiteboard that the image is still saved there

Test whole project: document the possible interactions with the GUI and use println to display what the GUI reads as the action and the response. We will run the GUI and tests each possibility.

- test one user one whiteboard--test every type of brush, every color, every size brush
- test one user multiple whiteboards: drawing on multiple whiteboards to make sure drawing are accurate
- test multiple user same board: drawing at the same time, drawing at different times, drawing at the same location at the same time, and using different brushes/colors/sizes all of these times
- test multiple users multiple whiteboards
- test someone drawing on a Whiteboard and someone erasing from the same spot on the same Whiteboard at the same time (indeterminate result???)

We are essentially trying to test every single aspect of our project, so this amount of effort should be enough. We are also testing all potential problems with each aspect of our project.

Testing GUI:

test location of drawing
test length of drawing
test erase function
test color change
test brush stroke size change
test drawing a line segment