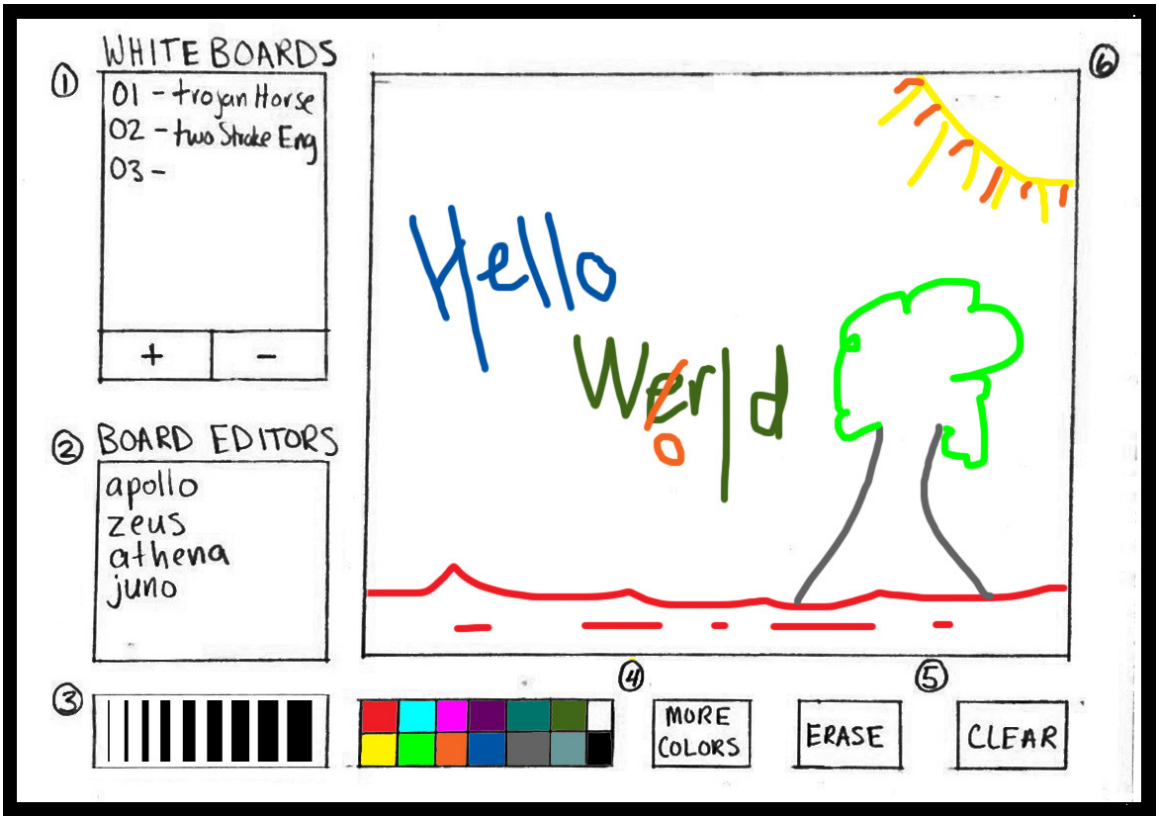
Design Milestone

Andre Aboulian, Cathleen Gendron, & Jon Beaulieu

6.005 Software Construction - Fall 2013 - Project 2: "Collaborative Whiteboard"

I. User Functionality Overview

I. Components



I.1 Selector

The board selector in the left pane includes a list of all current whiteboards and appears the same for all users. Each line represents an individual board, which are numbered sequentially and named by the user. Upon clicking the "+" button, the user will be prompted to name the new board. When the board has been created on the server, it will be appended to the list for each user. Selecting a board in the list will download the board from the server, overwrite the local copy if one exists, and display the board in the canvas window.

I.2 Board Editors

Displays a list of users, including the viewer, who are currently modifying the selected board. This list will be updated as users enter and exit the board.

I.3 Thickness Selector

This tool allows the user to select a brush/eraser thickness for drawing on the whiteboard.

I.4 Color Selector

The main color palette displays a grid of colors from which the user can choose to paint with. The color currently in use will be highlighted. Clicking the "more colors" button will open Swing's built-in color chooser, which will offer a larger selection of colors.

I.5 Erasing Tools

The erase button will allow the user to toggle between erasing and painting. "Erasing" will be defined as drawing with a white selection. Erasing will happen in the same order as drawing, so whichever request reaches the server first will erase all that has been drawn under it. Toggling back to painting will restore the user's previous color choice.

I.6 Whiteboard Window

Displays the currently selected whiteboard, including all of its drawn strokes and erasures. The whiteboard be real-time interactive to allow users to collaborate simultaneously. Edits will be made in the order that modifications reach the server. In other words, a stroke logged on the server at a specific instant will be drawn over any strokes drawn before that instant.

II. Behavior

Erasing Because the default canvas color is white, erasing will function the same as drawing with the color white selected. The thickness functionality that is present during normal drawing will continue to work with erasing. The Erase button on the client GUI will function as a toggle. Clicking the button once will switch to erasing mode, which essentially is simply switching the client's selected color to white. At this time, the user's previously selected color will be stored for future use. Clicking the button a second time will return the user's selected color to what is was before the user entered erasing mode.

Editing a Deleted Board Clients should not be allowed to edit a board that has been deleted by another user. To prevent this, users who are working on a board when it is marked for deletion by another user will be presented with a message box, stating that their board has been deleted. The deleted board then becomes unselected by any client GUIs who had the board selected at the time of deletion, forcing users to select a different whiteboard to work on. Simultaneously, the board will be removed from the master list of whiteboards available, so that no users in the future may accidentally select the deleted board.

II. Server-Client Communication

I. Protocol

I.1 Grammar

The following grammar will facilitate the text-based communication between the clients and the server. The server will send `StoC_MSG` messages to the client, which will be able to send `CtoS_MSG` messages back to the server.

```
StoC_MSG :== (STROKE | BRD_INFO | BRD_DEL | USER_INIT | BRD_USERS) N

CtoS_MSG :== (STROKE | SEL | BRD_REQ | BRD_DEL | BRD_ALL | USER_REQ) N

STROKE :== "stroke" S BOARD_ID S THICK S COORDS S COLOR
COORDS :== X1 S Y1 S X2 S Y2
X1, Y1, X2, Y2 :== INT
COLOR :== [0-255] S [0-255] S [0-255]
THICK :== [1-10]

SEL :== "select" S BOARD_ID

BRD_REQ :== "board_req" S NAME
BRD_ALL :== "board_all"
BRD_INFO :== "board" S BOARD_ID S NAME
BRD_DEL :== "del" S BOARD_ID
BRD_USERS :== "board_users" S BOARD_ID (S USER_NAME)+

USER_REQ :== "user_req" S USER_NAME
USER_INIT :== "you_are" S USER_NAME

NAME :== [^N]
USER_NAME :== [A-Za-z]([A-Za-z0-9]?)+
BOARD_ID :== INT

INT :== [0-9]+
N :== "\r?\n"
S :== " "
```

I.2 Definitions and Usage

Below is the definition for each of the messages that can be sent across the network. Note that particular messages warrant a response after processing, and those returned messages are included in the definition.

`STROKE` A `STROKE` message is sent from a client to the server upon drawing a line.

1. A `WhiteLine` object is created for the line from (`X1`,`Y1`) to (`X2`,`Y2`) with thickness `THICK` and color `COLOR`. It is then added to the `MasterBoard` corresponding to the provided `BOARD_ID`.
2. The `STROKE` message is forwarded to all clients who are currently editing the same board, so that the line can be incorporated onto their whiteboards.

`SEL` Upon selecting a different board, the client sends a `SEL` request to the server.

1. The server clears all stroke messages queued to update the client's whiteboard.
2. The server disassociates the client with the current whiteboard if one is assigned and associates the client with the requested board.
3. A `BRD_USERS` message is sent to all users of the requested board, with the addition of the new editor, to inform them of the change. Another `BRD_USERS` message is sent to all users of the previous board, with the omission of the removed editor, to inform them of the change.
4. A sequence of `STROKE` messages are sent to the client to recreate the current state of the selected whiteboard.

`BRD_REQ` When the client wants to create a new board, it send a `BRD_REQ` message with a properly formatted `NAME`.

1. The server initializes a new `MasterBoard` object with relevant properties and adds it to its list of boards.
2. A `BRD_INFO` message for the new board is sent to all connected clients to inform them of the newly available whiteboard. The clients add this board to their list of available boards.

`BRD_ALL` This is often called by the client upon initialization. It prompts a series of `BRD_INFO` replies for all existing boards.

`BRD_DEL` When the client wants to remove a board, it send a `BRD_DEL` message to the server.

1. If the provided `BOARD_ID` exists, the server removes the designated board from its central list of boards.
2. The `BRD_DEL` message is forwarded to all clients to signify that the board is no longer available. The clients remove this board from their list of available boards.

`USER_REQ` Upon entering a username in the client application, a `USER_REQ` message with a properly formatted `USER_NAME` will be sent to the server to request the desired username.

1. The requested username is checked against existing username. If the name already exists, a name is generated using the new board's ID number.
2. A `User` object with relevant properties is created to represent the new client.
3. A `USER_INIT` message is sent to the client to inform it of its acquired username.

Note that `BRD_USERS`, `BRD_INFO`, and `USER_INIT` are used solely as replies and are included in the descriptions above.

II. Data Transport

For a detailed explanation of the communication method between the clients and the server, please see the "Threads and Queues" section.

III. Data Type

I. `MasterBoard`

I.1 Description

`MasterBoard` represents a single whiteboard on the server. The instance is held by several `User` objects editing the board, which are mutually stored in the `MasterBoard` object; in other words, the board is aware of its editors and the editors are aware of the board.

I.2 Fields

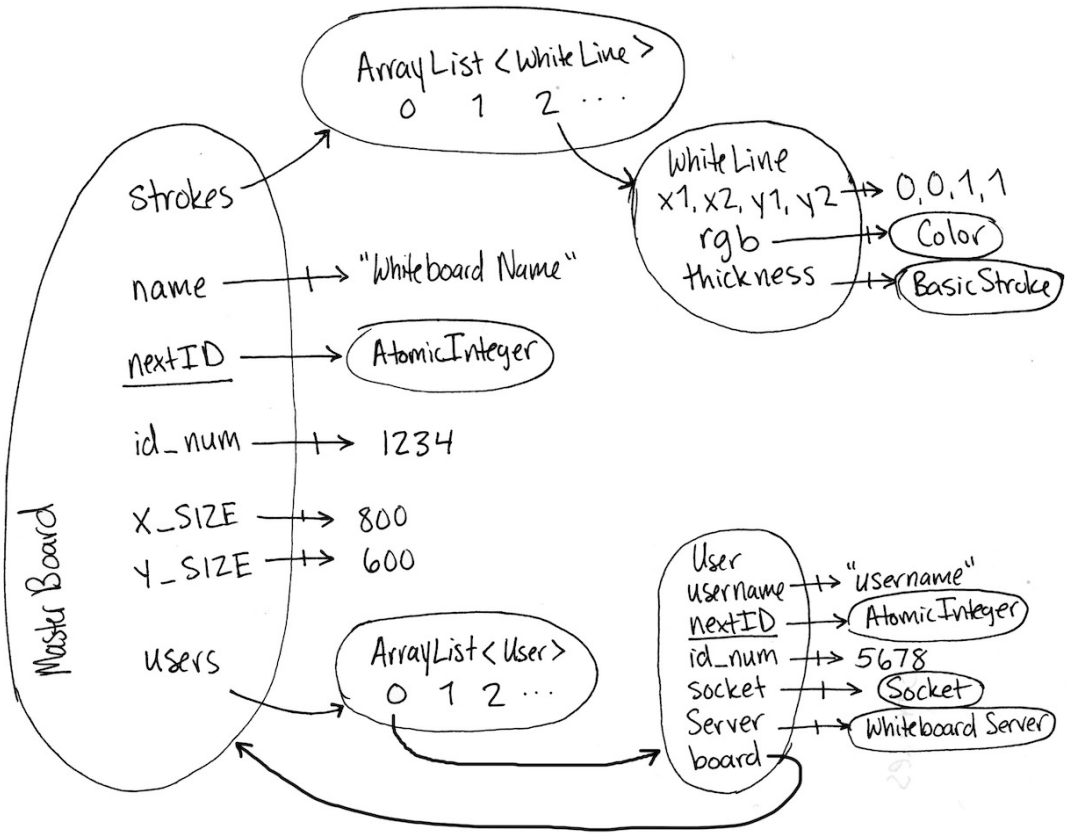
- `final String name` - The name of the board in the grammar format `NAME`.
- `final int id_num` - The unique identification number of the board.
- `static final AtomicInteger nextID` - The identification number of the successive board to be created.
- `final static int Y_SIZE, X_SIZE` - The X and Y dimensions of the whiteboard.
- `final ArrayList<WhiteLine> strokes` - A list of the strokes drawn on the whiteboard. The ordering of these strokes corresponds to the sequence in which modifications were made on the server and the layering of the segments.
- `final ArrayList<User> users` - A list of editors currently modifying the board.

I.3 Methods

- `WhiteLine[] allStrokes()` - Returns all the strokes that have been made on the board thus far by locking on `strokes` list.
- `void makeStroke(Stroke stroke)` - Adds the new `stroke` while locking on the `strokes` list. Proceeds to call `incorporateStroke()` on each user while locking on the `users` list.
- `void addUser(User user)` - Adds the new `user` while locking on the `users` list.
- `void removeUser(User user)` - Removes the user `user` while locking on the `users` list.
- `User[] getUsers()` - Returns all current editors while locking on the `users` list.
- `String getName()` - Returns the name of the whiteboard.
- `int getID()` - Returns the unique identification number of the board.
- `String toString()` - Returns the properties of the board in the form of a `BOARD_INFO` message.

- `boolean equals(Object other)` - Compares boards for equality on the basis of ID number.
- `int hashCode()` - Returns the identification number of board added to 1000.

I.4 In Action



II. `User`

II.1 Description

Each instance of `User` represents a separate client connected to the server. `User` is not only responsible for storing properties that pertain to the client; it also facilitates the socket communication and request handling for each client.

II.2 Fields

- `final Whiteboard Server` - The main server instance is stored for later global calls.

- `MasterBoard board` - The current board being edited by the user. A null value indicates that no board is selected.
- `final String username` - The unique acquired username assigned by the server after a username request has been made.
- `final int id_num` - The unique identification number of the user.
- `static final AtomicInteger nextID` - The identification number of the successive user to be created.
- `final Socket socket` - The socket corresponding to the client application represented by the `User` instance. The input and output streams are contained within.
- `BlockingQueue queue` - The queue containing all outgoing messages to the connected client.

II.3 Methods

- `void incorporateBoard(MasterBoard board)` - Queues a `BRD_INFO` message to be sent to the client with priority. This method is called by the main `WhiteboardServer` on all `User` instances once the request for a new board has been processed. `void forgetBoard(MasterBoard board)` - Queues a `BRD_DEL` message to be sent to the client with priority. This method is called by the main `WhiteboardServer` on all `User` instances once the request to remove a whiteboard has been processed.
- `void incorporateStroke(WhiteLine stroke)` - Queues a `STROKE` message to inform the client of a new stroke drawn on the current board.
- `void handleRequest()` - Runs in a background thread and takes care of all incoming requests from the client. Appropriate actions are taken for each request, either with the `server` or on the `board`.
- `void selectBoard(int boardID)` - This method is called upon receiving a `SEL` command from the client. The user removes itself from the current board by calling `removeUser(this)` on the locked `board`, replacing `board` with on acquired from `server.getBoard(boardID)` and calling `addUser(this)` on the new reference. All queued `STROKE` are removed since they pertain to the old board. The method finally calls `board.allStrokes()` to queue all previously drawn strokes for the new board to be sent to the client.
- `String getName()` - Returns the username of the connected client.
- `int getID()` - Returns the unique identification number of the connected client.
- `int compareTo(Object other)` - Compares users for ordering on the basis of usernames.
- `int equals(Object other)` - Compares users for equality on the basis of identification numbers.
- `int hashCode()` - Returns the identification number of the user added to 2000.
- `String toString()` - Returns the uername and ID number of the user in string format.

III. `WhiteLine`

III.1 Description

`WhiteLine` is a simple class designed to represent a drawn stroke on a whiteboard. It is used by the `MasterBoard` class on the server side and the `ClientView` class on the client side. Due to the frequent need to work with lines and the neatness of working with a single object containing the multiple properties of the line – location, color, thickness – we decided to have a dedicated class. The class is immutable, as all its properties are themselves immutable and stored upon construction.

III.2 Fields

- `final int x1, y1, x2, y2` - The coordinates of the line, which spans from `(x1,y1)` to `(x2,y2)`.
- `final java.awt.Color color` - The color of the line. The `Color` object allows greater versatility over storing individual RGB values, as Java provides multiple methods for specifying colors. This is especially useful for using a `JColorChooser`.
- `final java.awt.BasicStroke thickness` - The thickness of the line will be specified as a number within the range [1,10]. Numbers from this qualitative scale will be mapped to floating-point thicknesses and a `BasicStroke` object will be constructed internally.

III.3 Methods

- `int getX1()` - Returns the X-coordinate of the origin of the line.
- `int getY1()` - Returns the Y-coordinate of the origin of the line.
- `int getX2()` - Returns the X-coordinate of the terminus of the line.
- `int getY2()` - Returns the Y-coordinate of the terminus of the line.
- `java.awt.Color getColor()` - Returns the `Color` object corresponding to the line.
- `java.awt.BasicStroke getThickness()` - Returns the `BasicStroke` object corresponding to the thickness value specified upon construction.
- `String toString()` - Returns the properties of the stroke in the form of a `STROKE` message with the `BOARD_ID` omitted.

IV. `ClientBoard`

IV.1 Description

`ClientBoard` is used by the client application to store basic properties about the active whiteboards, which include those created by the client and other users. A new instance is created each time the client receives a `BRD_INFO` message. The class is immutable, as the contained properties are never changed after initialization on the server side.

IV.2 Fields

- `final String name` - The user-assigned name of the board.
- `final int id_num` - The identification number assigned to the whiteboard. This number is assigned by the server in the order that whiteboards are created; it cannot be the same for any two boards.

IV.3 Methods

- `String getName()` - Returns the name of the board.
- `int getID()` - Returns the identification number of the board.
- `int compareTo(Object other)` - Compares two boards on the basis of their ID numbers. Overrides the default comparison method and follows its conventions.
- `String toString()` - Returns the properties of the board in the form of a `BOARD_INFO` message.

V. `ClientView extends JPanel`

V.1 Description

`ClientView` has the simple task of managing the drawing canvas of the current board. It fulfills the role of the "View" component of the Model-View-Controller structure. Since the class that extends `JPanel`, it is used directly as a component within the client application.

V.2 Fields

- `final Image buffer` - The buffer where changes are made before being drawn to the displayed canvas.
- `final static int Y_SIZE, X_SIZE` - The X and Y dimensions of the whiteboard.

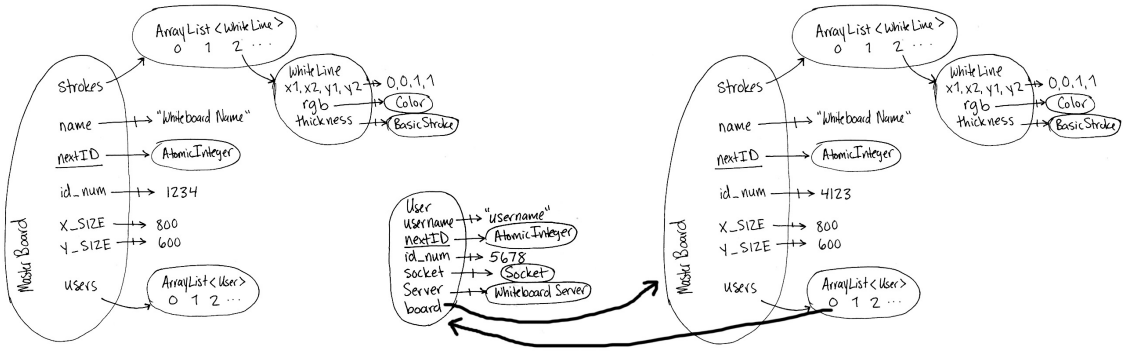
V.3 Methods

- `void drawLine(WhiteLine line)` - Draws the designated `line` on the `buffer`, which does not appear on the displayed canvas.
- `void clear()` - Clears the displayed canvas and the contents of the `buffer` immediately.
- `void push()` - Displays all changes that were made since the last time `push()` or `clear()` was called. Copies the contents of the `buffer` to the displayed canvas.

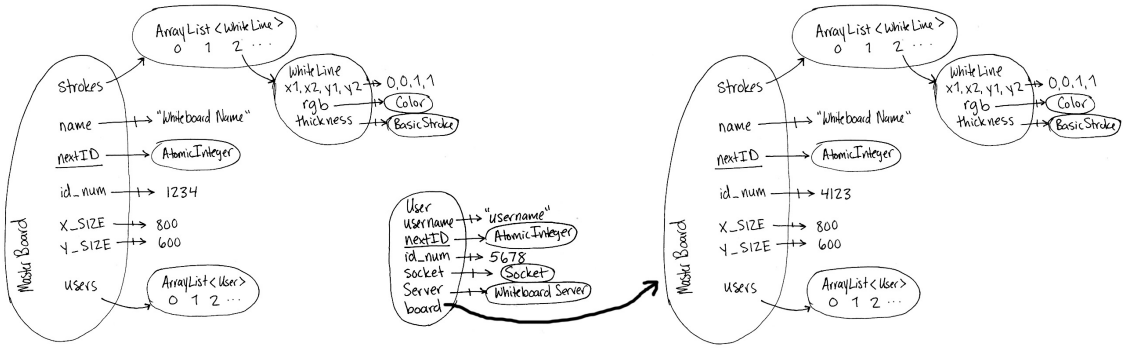
VI. Board Switching

The following sequence of steps defines the back-end operation of changing a client's whiteboard.

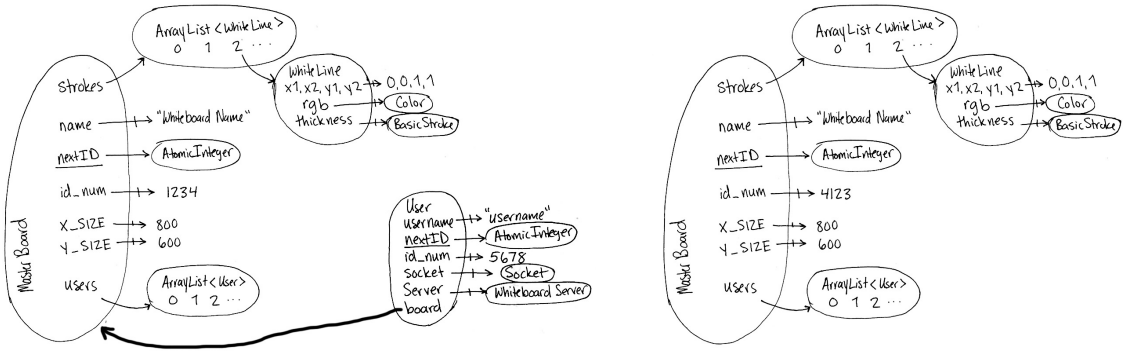
1. The `User` instance that represents a specific connected client begins with a reference to a `MasterBoard` instance. That `MasterBoard` instance has that associated `User` in its `users` list. The client then selects another board in the list of available boards, sending a `SEL` message to the server and is processed in the `User` class.



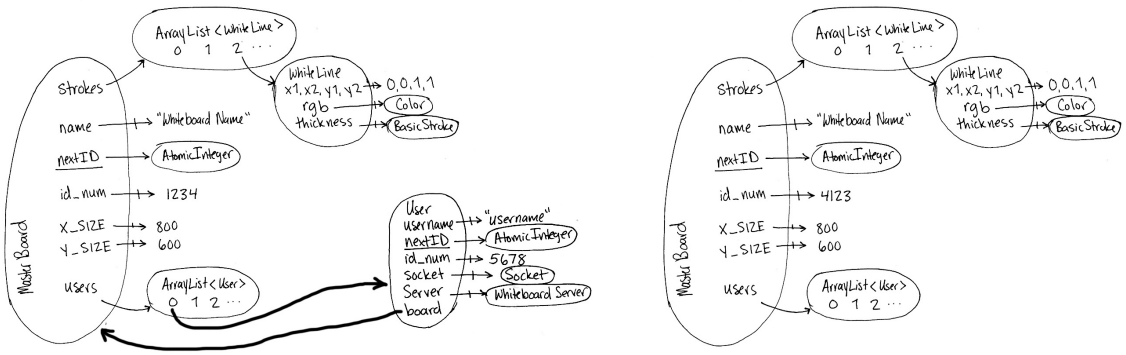
2. The `User` calls `board.removeUser(this)` on its current board, causing the board to no longer identify the client as one of its users.



3. The `User` executes `board = server.getBoard(id_num)` obtain the requested board from the server and reassign its reference to it.



4. The `User` calls `board.addUser(this)` on its new board, causing the board to identify the client as one of its users.



IV. Threads and Queues

Client The GUI maintains two threads, one which listens for requests from the associated `User`, and another which utilizes a buffered output stream to send message to the `User`. Possible requests include creating a new Whiteboard, deleting a Whiteboard, selecting a different Whiteboard, or sending a Stroke.

User Each `User` object holds the client-side socket. The User maintains two threads, one which listens for requests from either its MasterBoard or its GUI, and another which contains a BlockingQueue containing requests to be sent to the GUI. The request() method parses GUI requests and sends the appropriately formatted text protocol message to either the MasterBoard (when a Stroke is made) or the WhiteboardServer (for other User/Whiteboard requests).

MasterBoard The MasterBoard maintains two threads, one which listens for requests from any of its associated Users, puts the pending changes on a BlockingQueue, and processes these requests, and another which contains a BlockingQueue containing requests to be sent to all of its associated Users. For example, the listening thread will accept a request to create a new Stroke to be added to its strokes ArrayList. This update is then sent to the `User`s from the sending thread.

WhiteboardServer The WhiteboardServer is responsible for instantiating new Users with each new connection, and creating new MasterBoard objects in response to requests from Users. It maintains an InputStream to receive requests from the Users, but does not need an OutputStream.

V. Thread Safety

I. Processes

I.1 Adding New Users

Upon receiving a `USER_REQ` message, the WhiteboardServer calls `createNewUser` on the main server thread. This instantiates the new User object and adds it to the `users` ArrayList. This method locks on the `boards` field first, and then the `users` field. This ensures that only one User is created at a time, and that behavior that is dependent other User objects, such as username and ID assignment, is consistent.

I.2 Adding and Removing Boards

Upon receiving a `BRD_REQ` message, the WhiteboardServer calls `createBoard`, which instantiates the new MasterBoard object and adds it to the `boards` ArrayList. This method locks on the `boards` field first, and then the `users` field. If multiple requests for a new board are sent, the locks ensure that another board is not created until the current board is created, added to the ArrayList, and all Users are updated with these changes.

Upon receiving a `BRD_DEL` message, the WhiteboardServer calls `removeBoard`. This removes the board from the server's `boards` ArrayList, as well disassociating all Users currently active on that board. This method locks on the `boards` field and then the `users`, so that only one board can be deleted at a time, and so that one thread cannot be deleting a board from the ArrayList while another is attempting to add one.

I.3 Drawing Strokes

Each time a line is drawn on a client's Canvas, a request is sent from the WhiteboardClient to the MasterBoard via a buffer. The MasterBoard puts this message on a queue of pending requests, then locks on the `strokes` ArrayList and appends the stroke encoded in the message. The use of the threadsafe BlockingQueue ensures that all strokes are sent from the GUI to the MasterBoard in the correct order. Before releasing the lock, all Users associated with this board are informed of the newly added stroke.

I.4 Selecting Boards

When the client selects a new board, the server fetches the board to be sent. The User calls `allStrokes` to obtain all of the strokes associated with the new board. This method locks on the `strokes` ArrayList, so that no new strokes can be added before the User has received the entire ArrayList. The User also adds itself to the MasterBoard's `users` ArrayList, locking on that field to ensure that only one User is added at a time.

II. Averted Race Conditions

II.1 New Board and New User

Both `BRD_REQ` and `USER_REQ` messages are handled on the main server thread. Since only one of these requests can be processed at a time, there cannot be any racing between the two methods that would potentially cause a newly created board to be lost for some Users.

II.2 Remove Board and New User

Similar to the strategy in the previous section, both `DEL_BRD` and `USER_REQ` requests are handled on the main server thread. This eliminates any racing in which a new User would be created and still receive a board that had actually been deleted.

II.3 Concurrent Strokes

Because the GUI sends the new stroke messages through a BlockingQueue, the order is preserved. The MasterBoard then appends the strokes to its `strokes` in the order that they are received from the clients. (Note that this is not necessarily exactly the same as the order in which each stroke is sent from each GUI, but every User will receive strokes in the exact same order when receiving updates from the MasterBoard.) Because the `makeStrokes` method locks on `strokes` while it

processes a request, only one stroke will be processed at a time, and the risk of losing strokes or having clients receive stroke updates in different orders is eliminated. Similarly, while a newly selected board is being transferred to a client, no new strokes can be added, to eliminate these same risks.

II.4 Atomic ID Generation

All object IDs are generated by a static AtomicInteger associated with each class. Because the AtomicInteger is threadsafe, any risks associated with interleaving integer operations are eliminated.

III. Thread-Safe Collections

The server-side classes `User` and `MasterBoard` contain a number of ArrayList fields, referenced in the preceding sections. With the exception of `strokes` in `MasterBoard`, all of these fields will utilize the thread-safe synchronized wrapper, to add additional protection against concurrency problems, such as deadlock. The `strokes` field will not use this wrapper, because the `MasterBoard` uses a BlockingQueue for pending changes, which ensures that the `strokes` ArrayList is not accessed in multiple threads concurrently, and to avoid the synchronized wrapper's performance penalty on an object accessed so frequently.

VI. Testing

I. Constructor Testing

I.1 New Board

Valid Name Test the creation of a new whiteboard with a valid name. This is simple to conduct by simply clicking the "+" button under the list of whiteboards. The client should be prompted to enter a name. Enter a valid name that is not already in use. The server should create a new `MasterBoard` object and add it to the list of existing boards. This should be seen by all clients. The new whiteboard should be completely white.

Invalid Name Test the creation of a new whiteboard with an invalid name. Again, conduct by simply clicking the "+" button under the list of whiteboards. The client should be prompted to enter a name. Enter a name that is already in use. The server should inform the user that the name is already in use, then create a new `MasterBoard` object with a default name assigned by the server. Again, this new `MasterBoard` should be added to the list of existing boards, and should be seen by all users. The new whiteboard should be completely white.

I.2 New User

Valid Name Connect to the whiteboard server. When prompted for a username, enter a valid one - that is, one that is not already in use. The server should accept this name and assign it to the user, then allow the user to select a board and begin drawing.

Invalid Name Connect to the whiteboard server. When prompted for a username, enter a name that is already in use. The server should notify the new client that his/her choice of username is already in use, and should assign the new user a different (valid) name from a predefined list of names. The new client should then be allowed to select a board and begin drawing.

II. Normal Functionality

II.1 Loading a Board

Tests behavior when a user attempts to load an existing board to work on. This will happen when a client clicks on a board name from the list of existing board on the left of the GUI. Expected behavior: The `User` associated with this client will be removed from the list of users working on the previous board. It will then be added to the list of users working on the newly selected board. This will be reflected on the list of users on the bottom left of the GUI for anyone working on the newly selected board. In addition, the client who changed selections will have his drawing space refreshed to contain an up-to-date version of his/her newly selected board. The client will then be able to modify this board as usual.

II.2 Creating a New Board

Tests behavior when a client attempts to create a new multi-user whiteboard. This event will occur when the client clicks the "+" button located below the list of existing whiteboards. When this button is clicked, the client will be prompted to enter a name for the new whiteboard. If the name is not valid (already in use), the server will assign a default one. The new whiteboard will appear in the list of existing whiteboards in the GUI, so that all users are able to access it. The client who created the whiteboard will automatically have his GUI switched so that he is viewing the newly created board, in accordance to the "Loading a Board" test above. The newly created `MasterBoard` will begin as a blank white canvas, with the standard 800x600 pixel dimensions.

II.3 Deleting a Board

Tests behavior when a client deletes an existing board. This will occur when a client clicks the "-" button located below the list of existing whiteboards. To prevent clients from accidentally (or purposefully) deleting others' work, the "-" button will always delete the whiteboard that the client is currently viewing. In addition, it will be impossible to delete a whiteboard if only one is currently is existence, to prevents errors arising from a lack of whiteboards. Deletion will completely remove the selected `MasterBoard` object from the server memory. It will be reflected in the list of whiteboards that appears in the user GUI, where the name of the deleted whiteboard will be removed for all users. Other clients who are working on a whiteboard when it is deleted will be notified via a pop-up message that their board has been deleted. Their workspace will then not have any whiteboard selected, meaning that they will need to load another whiteboard before continuing to draw. All requests sent to the server referencing the deleted whiteboard will be ignored after board deletion.

II.4 Choosing Stroke Thickness

This will test the user's ability to select a stroke thickness for drawing purposes. Thickness will be controlled through the `Width` component of a `Java 2D Stroke`. A variety of stroke widths will be available for selection from the thickness panel near the bottom left corner of the GUI. Each image in this area will be associated with a specified stroke thickness. Clicking on one of these images will switch the user's default stroke thickness to the width associated with the selected image.

II.5 Choosing Stroke Color

This will test the user's ability to select a drawing color. Thickness will be controlled through the `Color` component of `Java 2D Graphics`. A variety of colors will be available for selection from the color panel along the bottom of the GUI. Clicking on one of these given colors will switch the user's default stroke color to the selected color.

II.6 "More Colors" Button

In addition to choosing a color as listed above, users should be able to select from a wider range of colors in a `JColorChooser` window. This window will appear when a client clicks the "more colors" button to the right of the GUI's color panel. The client will then be able to change his default stroke color to any color provided in the `JColorChooser` window. If the user clicks outside of the `JColorChooser` window while it is open, the window will be closed and the client's color will not be changed.

II.7 "Erase" Button

The erase button will function as a toggle, switching the client between his previously chosen color and white. As the default board color is white, painting with white will appear visually the same as erasing an existing portion of the whiteboard. When the user selects "erase" for the first time, his/her color choice will switch to white. Clicking the "erase" button a second time will switch the user back to whichever color was selected before "erase" was clicked the first time.

II.8 "Clear" Button

The "clear" button will completely erase the client's currently selected board. On the server side, this will delete all strokes that have ever been applied to the selected whiteboard. As a result, the whiteboard will appear as it did when it was first created - blank white. This change, like any other painting operation, will be seen by all clients viewing the whiteboard.

II.9 Drawing on a Whiteboard

Whiteboard Selected When a client clicks and drags across the whiteboard area, a new instance of a `Stroke` object will be created to match the users' input with regards to length, shape, location, color, etc. This `Stroke` will be sent to the server, where it will be associated with the whiteboard it should be applied to. The server will then update the views of all other users working on the same whiteboard to include the new stroke. In this way, all users will be able to see all other users' changes in real time

No Whiteboard Selected If the client has not selected a whiteboard, attempting to interact with the canvas will result in a message box appearing, warning the user to select a whiteboard before drawing. The user's attempts to draw will produce no visible result on the canvas, nor will there be anything sent to the server.

II.10 Concurrent Board Operations

User A Deletes while User B Modifies The selected `MasterBoard` instance will be deleted. Both users (as well as any other users on the board) will be notified via message box that it has been deleted. They will all then need to select another whiteboard in order to continue drawing. User B's request to the server will be ignored.

Switch then Disconnect Test when a user switches boards, then immediately disconnects from the server. The user's name should disappear from all lists, including the board he/she switched to just before disconnecting. All other users should see this in the list of users currently editing whiteboards.

Rapid-Switching between Boards Test when a client switches very quickly between a number of boards. The server should only attempt to update the client's view to the most recent whiteboard selected - that is, it should abandon any attempts to update the client's board to a previously selected board, even if the loading process is not complete. At the same time, the client's GUI should ignore any server messages regarding a whiteboard that is no longer selected. In this way, the client will only ever see their most recent whiteboard selection loading.

Edit then Switch Test when a user makes a stroke on a whiteboard, then very quickly switches to another board. The client's edits should still be sent to the server, and be reflected on the server's copy of that board, so that all other users can see the edit. The client who made the edit, however, should not have it reflected back to him/her by the server. Instead, the client should go through the normal process of loading their newly-selected board.

II.11 User Disconnect

Test the normal functionality of a user disconnecting from the server. The user's GUI client should close, and the user's name should be removed from the whiteboard it was currently working on, meaning that it should not appear anywhere on the server. All other clients should be able to see this change.