6.005 Pandaboard Revised Design Document

Austin Freel, Michael Handley, Johannes Norheim

Major Changes

The high-level design pattern for this project underwent minimal changes. Our main classes (WhiteboardGUI, WhiteboardModel, WhiteboardServer, Whiteboard, and Client) all remain, and play exactly the role we expected them to. We added capability for more messages to our protocol; added some extra classes for the client-side GUI; and constructed an interface for our whiteboard front end. The interface makes WhiteboardModel independent from our GUI implementation, which makes our code ready for change. If somebody wanted to generate a GUI, say for Android, they only need to implement the methods specified in the interface in order to rely on the model. It also allows us to easily do automated testing with the model by making a dummy GUI class that implements every method in the interface without having to build the GUI.

The most significant changes which came about were the GUI changes, and the changes that had to be made because of them. We opted to have separate phases for our GUI, with the content on the GUIs lower bar changing depending on the input required from the user (initially fields for IP and port of server, then fields for username and whiteboard, then just a whiteboard menu). This meant that we had to further modularize our code by adding in methods like WhiteboardModel#ConnectToServer and WhiteboardModel#ConnectToWhiteboard, which saved us from having a WhiteboardModel instance connect to the server on its construction. After our initial connection, however, messages from the server are all still handled by our ServerListener Runnable class.

Other more significant changes were caused by our addition of a feature which colors users' names in the users list displayed on WhiteboardGUI in the same color as the line they are drawing. This required modification of our "line … " message as well as insurance that usernames are unique. To ensure this, we add and keep track of a usernames list in WhiteboardServer.

Message Protocol:

To our message protocol, we added some new possible messages - a "disconnect" message; a "usernameTaken" message; a "newUser" message; and a "removeUser" message.

- The "newUser <user>" and "removeUser <user>" messages are sent to clients, and were added in order to make our coloring of a user's name when that user draws work smoothly. These changes also had the effect of making the update operation on a connected user for when a new user connects to their whiteboard more lightweight (as the full list of users is no longer sent, and then reloaded client-side) as well as increasing the modularity of our code further down the line (UsersBar#loadUsersBar now simply

makes repeated calls to UsersBar#addUser).

- The “disconnect <user>” message is sent to the server when a client chooses to close their GUI, and notifies the server that it should remove the client from the server-side Whiteboard it was stored in. This was a necessary functionality that we did not catch in our initial design draft, but which required no significant structural change to our datatype.

- The “usernameTaken” message is sent to a client from the server when that client calls Whiteboard#connectToWhiteboard with a username that is used by another client connected to the server. On receipt of this message, the client’s GUI updates by loading a .bmp file telling the user that they must enter in a different username.

Additionally, we modified our “line … “ message so that it now includes the username of the client who drew the line. This was done to allow an additional feature that we decided to include: when a user draws a line in a certain color, their username on the GUI of each client connected to their whiteboard changes color to the color they are drawing in.

Classes:

Three new classes were added: UsersBar, TopButtonBar, and BottomButtonBar. They were added in order to enrich and further modularize our GUI. They don’t cause any structural change to our datatype. Events in these boards are listened for in WhiteboardGUI.

- UsersBar:
    - This class is used to display all the users connected to the same whiteboard that our client is connected to.
    - Takes in a list of current users as an argument to its constructor, and calls loadUsersBar to add to itself JLabels of each user.
    - loadUsersBar(List<String> users) is called only when this client connects for the first time (when this UsersBar instance is instantiated), and makes consecutive calls to addNewUser for each user in users in order to populate this UsersBar.
    - addNewUser(String user) updates this UsersBar by adding to it a new JLabel of user.
    - removeUser(String user) is called when a user disconnects from the whiteboard this user is connected to. Removes that user’s JLabel from this UsersBar.
    - updateUserColor(String user, int r, int g, int b) is called when a “line ..” message is received. Changes the color user’s JLabel on this UsersBar to that specified by the RGB components passed in.
- BottomButtonBar:
    - Holds forms for input related to whiteboard connectivity. Has three states:
        - Before a client connects to the server (has fields for the IP and port of the server, and a “Connect!” button).
        - After connection to server has been established but before a username and whiteboard have been specified (has a field for the client’s username, a menu for whiteboard selection, and a “Join Board” button).

        - After joining a whiteboard (has a board menu and “Join Board” button)
    - Transitions through these states are handled in WhiteboardGUI, using BottomButtonBar#add(JComponent comp) and BottomButtonBar#remove(JComponent comp).
    - Listeners for the components held on a BottomButtonBar instance are added in WhiteboardGUI.
- TopButtonBar:
    - Holds controls for various drawing features on the client’s WhiteboardGUI instance:
        - An image of a pencil/eraser when the client is drawing/erasing, respectively.
        - An “Eraser button”.
        - A “Choose Color” button, which displays a JColorChooser palette when clicked.
        - A slider for the user to select the size of line draw.
        - A “Save Image” button, which saves the current state of the whiteboard locally as a .bmp file when clicked.
    - Listeners for components held on a TopButtonBar instance are added in WhiteboardGUI.

Interfaces:

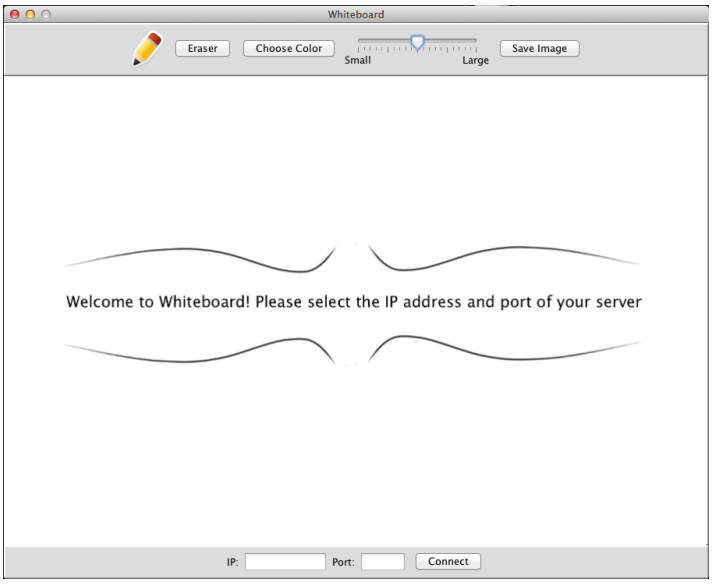
- WhiteboardFrontEnd:
    - Requires all methods related to our message protocol:
    - drawLineOnGui(String x1, String y1, String x2, String y2, String width, String r, String g, String b)
        - draws the line on the gui’s “canvas”(or similar display)
    - addNewUser(String user)
        - adds a new user to a display list on the gui
    - removeUser(String user)
        - removes user from the gui display list
    - loadGuiUsers(List<String> usersList)
        - loads all users in a list on the gui
    - loadUsernameTakenImage()
        - tells the client that the username has been taken, preferably by loading the warning on the screen.
    - loadConnectedToServerImage()
        - first image that draws on canvas when the client connects

Our classes consist of, on the client side, WhiteboardGUI and WhiteboardModel, and, on the server side, WhiteboardServer, Whiteboard, and Client. Further, our WhiteboardServer has a subclass clientHandler that implements Runnable. WhiteboardGUI is an extension of the Canvas class provided for us, and represents the GUI of the user, as diagrammed below. The WhiteboardModel is the part of the client that waits for messages from the server and then passes these messages on to WhiteboardGUI methods that will handle them (i.e. drawing lines

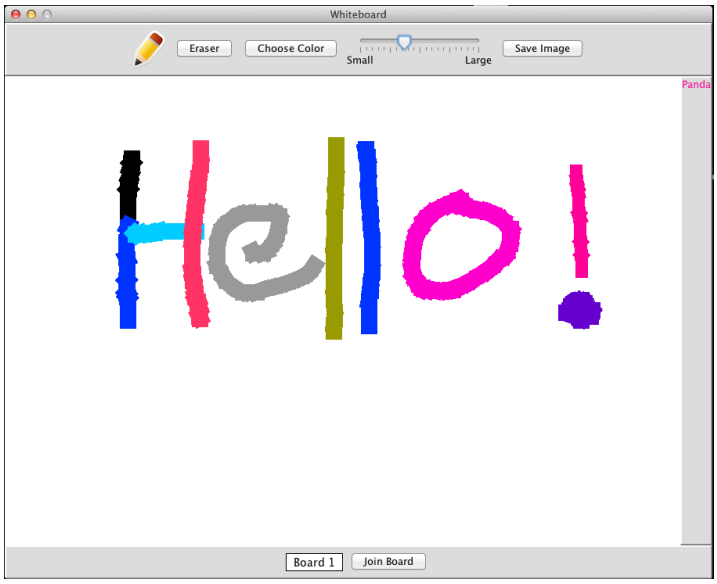
on a user's local whiteboard). When a client connects to the server, an instance of Client using that client's credentials is created on the server side and stored for future reference. The class Whiteboard represents a whiteboard that all clients are connected to. It is responsible for storing all the details of a specific whiteboard on the server and subsequently updating clients that are using that whiteboard (it is also updated by clients that are working on this whiteboard instance), as well as pushing the most recent vesion of the whiteboard to new connecting users. WhiteboardServer, along with clientHandler, deal with the connection of new users on the server side as well as any incoming messages from these users

Client Side

WhiteboardGUI



---->



NOTE: the following are added to the GUI in the main method

- TopButtonBar topbar - northern display bar on GUI (see above for class details)
- BottomButtonBar bottombar - southern display bar on GUI (see above for class details)
- UsersBar usersbar - eastern user display bar on GUI (see above for class details)

Methods:

- Extension of the pre-given class Canvas.
- Listeners(Functors) call *model*.sendMessageToServer() when the client wants to connect to a whiteboard or draw a line.
- loadWelcomeImage()
    - Loads the WelcomeImage.bmp file to the user's canvas. Called on GUI inititalization.
- loadConnectedToServerImage()
    - Loads the ConnectedToServerImage.bmp file to the user's canvas. Called once client successfully connects to the server and must now enter their username and desired whiteboard.

- loadUsernameTakenImage()
    - Loads UsernameTakenImage.bmp to the user's canvas. Called when the client attempts to use a username already stored on server.
- paintComponent(Graphics g)
    - Makes our canvas and draws it to the GUI
- drawLineOnGUI(String strx1, String stry1, String strx2, String stry2, String strwidth, String strr, String strg, String strb, String user)
    - Draws a line on the local GUI at specified coordinates.
    - Calls usersbar#updateUserColor.
- loadGuiUsers
    - loads user list onto GUI user bar
- addNewUser
    - adds new user to GUI user bar
- removeUser
    - removes user from GUI user bar

WhiteboardModel

Methods:

- constructor(String host, int port)
    - Connects to a whiteboard server located on host port
    - Spins a thread that constantly listens for messages from the server(a while loop constantly running checking if there is any line in the inputStream)
- sendMessageToServer(String message)
    - Sends a message to the server through the output stream.
- disconnectFromServer()
    - Tells server client is disconnecting, called on closing of client's GUI window.
- connectToWhiteboard(String whiteboard, String username, boolean usernameConfirmed)
    - Attempts to connect to whiteboard with given username. If usernameConfirmed then only sends "whiteboard …" message to the server, if not then sends message and listens for server to approve username as unique.
- drawLineOnServer(int x1, int y1, int x2, int y2, int width, int r, int g, int b)
    - Sends "line .. " message to the server

Server Side

WhiteboardServer

Methods:

- WhiteboardServer()*:*
  - Waits for clients to connect to the server.
  - Starts a new thread for each connected client.
    - That thread waits for the user to select their desired whiteboard and username, and creates a new Client object on receipt of those values.
    - That Client object is then added to the specified Whiteboard instance.
    - Spins new thread of a clientHandler instance, passing in user's socket and whiteboard ID.

Classes:

- clientHandler(Socket socket, String whiteboardID, Client client) implements Runnable
  - Processes all messages sent through the socket, calling whiteboardMap.get(whiteboardID).addLine(String message)

Whiteboard extends JPanel

Methods:

- addClient(Client c)
  - locks on *clients*
  - sends message to every client in this whiteboard's client list *clients* to inform them of the name of the new user *c* that has connected
  - adds the new client *c* to the whiteboard's client list *clients*
  - sends this client *c* a message including the names of all users currently working on this whiteboard
  - sends this client *c* a list of of this whiteboards' history
  - releases lock on *clients*
- removeClient(Client c)
  - locks on *clients*
  - removes the new client *c* from the whiteboard's client list *clients*
  - sends message to every client in this whiteboard's client list *clients* to inform them of the name of the user *c* that has disconnected from this whiteboard
  - releases lock on *clients*
- addLine(String message)
  - locks on *history*
  - adds line message to history list
  - releases lock on *history*
  - sends this line message to all users currently connected to this whiteboard

Client

Fields:

- String username
- PrintWriter printWriter

Methods:

- Client(String username, PrintWriter printWriter)
  - Initializes Client instance using given values
- sendMessage(String message)
  - lock on this Client
  - Sends a message over the output stream to the socket associated with this client
  - Release lock on this Client
- getUsername()
  - returns this.username

Communication *(message)*

Message protocol: messages are strings and can be of three types:

1. "whiteboard x username y", where x is the id of the whiteboard the client wants to connect to, and y is the user's chosen username. This message is only generated by the Client and sent to the Server.
2. "users a b c … n", where a, b, c, etc. are the usernames of all the clients that are currently interacting with a specific whiteboard. This message gets sent by the server to a new client so that he may populate his list of current users working on that specific board.
3. "line x1 y1 x2 y2 width r g b username", where x1, y1, x2, y2 are numbers describing the start and end points of a line; width is a number describing the line length; r, g, b are values describing the line color; and username is the name of the user drawing the line. This message will be generated by a client when that client draws a line on the whiteboard. The message will be passed to the server and subsequently sent to all the users that are currently interacting with that specific board. That line will then be drawn locally on the board for each client, including the user who generated the line originally. The username is passed in so that the user bar may also be updated as to show each user's current color by coloring their name that color on each clients' GUI.
4. "newUser username", where username is the name of the client being added. This message gets sent to all clients connected to the same whiteboard as the new user so that they may update their user lists and corresponding user colors.
5. "removeUser username", where username is the name of the client being removed. This message gets sent to all clients connected to the same whiteboard as the existing user left so that they may update their user lists to no longer contain this user.
6. "disconnect username", where username is the name of the client closing his GUI. This notifies the server that it should remove the client from the server-side Whiteboard it was stored in.
7. "usernameTaken" message is sent to a client from the server when that client attempts to connect to the system with a username that is being used by another client. On receipt of this message, the client's GUI updates by loading a .bmp file telling the user that they must enter in a different username.

Message usage: Messages are passed through the input and output streams.

Concurrency Strategy

Fields which may threaten thread safety:

- Whiteboard.*clients*
- Whiteboard.*history*
- Whiteboard.*bmp*
- Whiteboard.*drawingBuffer*

- Whiteboard.*clients* is accessed by only Whiteboard#getClients, Whiteboard#addClient, Whiteboard#removeClient, and Whiteboard#sendMessageToAll. All of these methods must acquire a lock on *clients*, so *clients* is therefore protected by the Monitor Pattern. This prevents any race conditions which may have caused a client List to be returned which lacked a new client or removed client, or a new client to miss a message which was to be sent out to all.
- Whiteboard.*history* is accessed only by Whiteboard#sendHistory, Whiteboard#addLine, and potentially subsequent calls to Whiteboard#checkHistory, and Whiteboard#resetHistory. Both checkHistory and resetHistory are only called within addLine, which itself is synchronized on *history*. This ensures that, throughout the whole potential process of a line event being added, *history* being checked for its size, and *history* being reset and a new .bmp file created, there will be no changes to *history* from any other threads. Additonally, sendHistory also locks on *history*. Thus, by the Monitor Pattern, *history* is threadsafe.
- Whiteboard.*bmp* is accessed by only Whiteboard#sendBMP and Whiteboard#resetHistory. Competing calls of these two methods will not be an issue, as resetHistory is necessarily called through addLine, which means any event which caused a new .bmp to be created in resetHistory will already be queued for drawing on the whiteboard of any new client who would have incurred the sendBMP call. Thus an outdated BMP send will not be harmful as the recent event is still queued to the client. Another possible race condition is where the user connects and is sent a line message before it loads the new .bmp, draws that line, and then overwrites it with the old .bmp. This is avoided as both Whiteboard#sendMessageToAll and Whiteboard#addClient are locked on *clients*. This lock on clients therefore prevents any concurrency issues with our .bmp access as it prevents overwriting on the client's whiteboard.
- Whiteboard.*drawingBuffer* is threadsafe as it is accessed by only two methods, Whiteboard#drawLine and Whiteboard#resetHistory. As noted above, resetHistory is only called from checkHistory, which is in turn only called from addLine. drawLine is also only called from addLine. As addLine requires a lock on *history*, there will be no concurrency issues regarding these two method calls. Thus *drawingBuffer* does not threaten our thread-safety.

Client-side, commands to draw on a client's whiteboard are all received by the client through a single thread, so our client-side WhiteboardGUI is therefore threadsafe, as only a single thread mutates it. WhiteboardModel has one thread to listen for changes sent to it by the server, and one thread to send changes to the server. These two threads act in parallel, however, never crossing paths or affecting the same objects or values. More, WhiteboardModel has no fields or state to change, so it is also threadsafe.

Testing:

Partition of Input Space:

- WhiteboardGUI
  - Pre-connection
  - Post-connection
  - Listeners
- WhiteboardModel
  - sendMessageToServer(String message) method
- WhiteboardServer
  - client connecting to a chosen whiteboard
  - clientHandler(Socket socket, String whiteboardID)
- Whiteboard
  - addClient(Client c)
  - removeClient(Client c)
  - addLine(String message)
- Client
  - sendMessage(String message)

Coverage:

Client Side (note: these tests only ensure client side is working independently)

- WhiteboardGUI
  - Pre-Connection:
    - simple opening of GUI
    - loadWelcomeImage()
    - ensure cannot draw
  - Post-Connection:
    - loadConnctedToServerImage()
    - loadUsernameTakenImage()
    - ensure cannot draw (until successfully connected to whiteboard)
    - Listeners:
      - eraser
      - save image
      - input name
      - input IP
      - input port
      - board #
      - connect
- WhiteboardModel
  - sendMessageToServer(String message):
    - check that method runs without error (we test for receiving messages later)

Server Side (note: these tests only ensure server side is working independently)

- WhiteboardServer
  - when a new client connects, check to see that the Whiteboard object that the client connected to now contains the client (JUnit)
  - clientHandler(Socket socket, int whiteboardID)
    - testing when socket is valid/invalid
    - testing when whiteboardID is valid/invalid
    - processing of incoming messages is tested by sending messages to the server and ensuring that the server handles them and continues to run (we test for specific types of messages later)
- Whiteboard
  - addClient(Client c)
    - ensure client is added to the clients list (JUnit)
    - test case when clone of Client c already exists (JUnit)
  - removeClient(Client c)
    - ensure client is removed from the clients list (JUnit)
    - test case when Client c is not in the clients list (JUnit)
  - addLine(String message)
    - check that line drawn on GUI is correct
    - check that line is added to history list (JUnit)
- Client
  - sendMessage(String message)
    - ensure runs properly (JUnit)

Testing Over Connection (note: these tests are for interaction between client and server)

- Note that this process involves three main steps: (1) testing client side functionality with a dummy server, (2) testing server side functionality with dummy clients, and (3) finally testing interaction between our actual client and actual server.
  - (1) We will create a dummy server that will read all types of messages from the client and simply print them to output so we know that the client side can successfully connect to a server and communicate with it properly.
  - (2) We will create dummy clients that, like the dummy server, will read all types of messages from the server and print them to output so that we know the server can successfully connect and interact with specific clients.
  - (3) Combining steps (1) and (2), we will perform a full-functionality test between our client and server sides, checking to make sure that messages are sent and received correctly, and that they call the correct methods (which, at this point, have already been tested individually). The specificity of this testing step is described below.

- Connecting to Whiteboard
    - We will manually test that when a client connects to the server and chooses a whiteboard:
        - on the server side, this whiteboard object will have an updated client list
            - this subsequently tests our handling of the String message that is sent from client to server of the form “newuser x”
        - on the client side, all clients interacting with that whiteboard will have a client list of current users that will be updated to contain the new user
            - this subsequently tests whiteboard’s sendMessageToAll method
        - the client that connects will see a whiteboard that is up-to-date
            - this subsequently tests Whiteboard’s sendBMP, sendHistory, and loadWhiteboard methods
- Sending and Receiving Messages
    - We will manually test that when:
        - a client draws a line on a whiteboard:
            - a message is generated and sent to the server (checking that the respective whiteboard history is updated)
                - this tests the server’s handling of the String message “line x1 y1 x2 y2 width r g b”
            - the message is sent to all clients using that whiteboard and no other users
                - this subsequently tests whiteboard’s sendMessageToAll method