12/3/2013

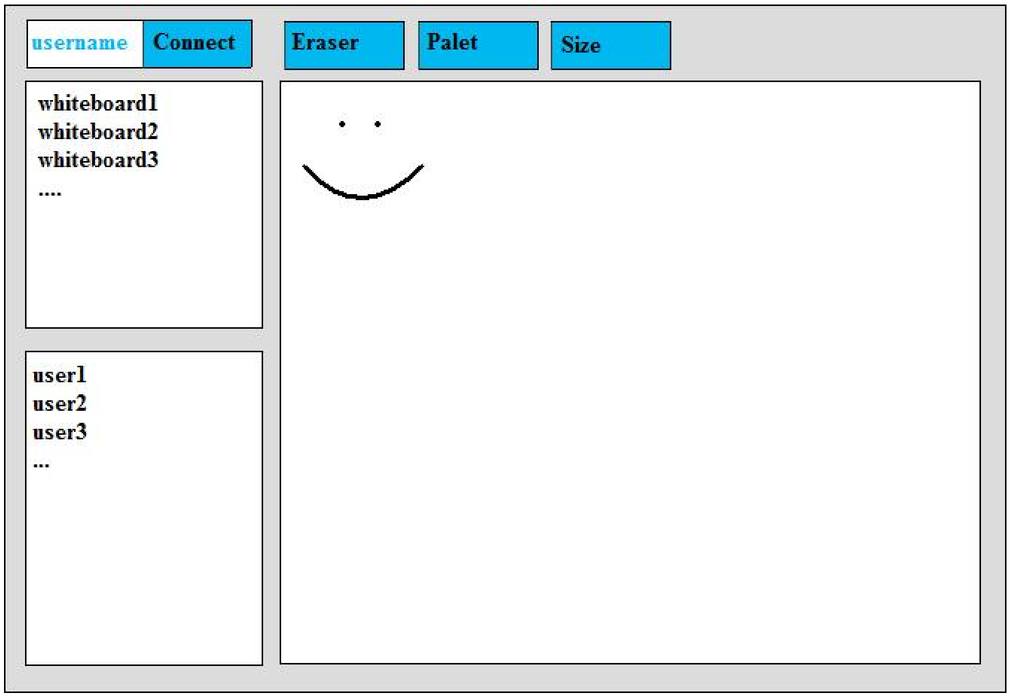
6.005 Pandaboard Design Document

Austin Freel, Michael Handley, Johannes Norheim

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



Fields:

- WhiteboardModel model
- List users

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.
- doAction(String message):
    - if message is “newuser x” adds the user to the GUI list of users
    - if message is “line x1 y1 x2 y2 width r g b” draw the line on the gui

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.

Server Side

WhiteboardServer

Fields:

- HashMap<Integer, Whiteboard> whiteboardMap

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

Whiteboard extends JPanel

Field:

- List<Client> clients
- List<String> history
- File bmp BMP file representing our last saved image
- Image drawingBuffer representing the current GUI image

Methods:

- getClients()
    - locks on *clients*
    - returns the list of clients

    - releases lock on *clients*
- addClient(Client c)
    - locks on *clients*
    - adds a new client to the whiteboard
        - send BMP file to Client
        - add Client to *clients*
    - sendMessageToAll("newuser " + c.getUsername())
    - loadWhiteboard(c)
    - releases lock on *clients*
- removeClient(Client c)
    - locks on *clients*
    - removes a client from the whiteboard
    - sendMessageToAll("removeuser " + c.getUsername())
    - releases lock on *clients*
- sendMessageToAll(String message)
    - locks on *clients*
    - Client.sendMessage(String) for each Client in *clients*
    - releases lock on *clients*
- drawLine(x1, y1, x2, y2, width, r, g, b)
    - draws line onto *drawingBuffer*
- resetHistory()
    - creates new BMP from *drawingBuffer*, and sets BMP file to this
    - sets history list of line messages to be empty
- checkHistory()
    - if history list is above certain size: resetHistory()
    - else: do nothing
- addLine(String message)
    - locks on *history*
    - adds line to history list
    - parses message and calls drawLine
    - checkHistory()
    - releases lock on *history*
    - sendMessageToAll(message)
- sendBMP(File bmp, Client c)
    - Sends the most recent .bmp file of this whiteboard to the client
- sendHistory(List<String> history, Client c)
    - locks on *history*
    - Send each item in the history list to the client.
    - releases lock on *history*
- loadWhiteboard(Client c)
    - Loads current state of this whiteboard onto c's WhiteboardGUI using this whiteboard's history list and latest BMP.
    - sendBMP(this.bmp)

- sendHistory(this.history)

Client

Fields:

- String username
- Socket socket

Methods:

- Client(String username, Socket socket)
    - 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. "newuser x", where x is the username of the new user. This message gets sent by the server to all of the clients that are currently working on the same board as x so that they may update the list of current users working on that specific board.
3. "removeuser x", where x is the username of a disconnected user. This message will be sent to all clients connected to the whiteboard that user x chose to disconnect from. On receipt of this message, clients' gui will be updated to reflect the loss of user x.
4. "line x1 y1 x2 y2 width r g b", where x1, y1, x2, y2 are numbers describing the start and end points of a line, width is a number describing the line length, and r, g, b are values describing the line color. 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.

Flexibility: The types of messages can eventually be extended to support other types of objects, e.g.: "circle x y r".

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

Concurrency Strategy

Fields which may threaten threadsafety:

- 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
  - Listeners
  - doAction(String message) method
- WhiteboardModel
  - sendMessageToServer(String message) method
- WhiteboardServer
  - client connecting to a chosen whiteboard
  - clientHandler(Socket socket, int whiteboardID)
- Whiteboard
  - getClients()
  - addClient(Client c)
  - removeClient(Client c)
  - sendMessageToAll(String message)
  - drawLine(x1, y1, x2, y2, width, r, g, b)
  - resetHistory()
  - checkHistory()
  - addLine(String message)
  - sendBMP(File bmp, Client c)
  - sendHistory(List<String> history)
  - loadWhiteboard(Client c)
- Client
  - sendMessage(String message)

Coverage:

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

- WhiteboardGUI
  - test to make sure Listeners correctly activate and run methods within them
  - doAction(String message)
    - if message is “newuser x”, adds the user to the GUI list of users (JUnit)
      - test also if x is already a user
    - if message is “line x1 y1 x2 y2 width r g b”, observe it draws the line on the whiteboard canvas
      - edge case values of all integers:
        - x and y values: valid, 0, equal, invalid (negative or too large)
        - width: negative, 0, positive
        - r,g,b: invalid and valid integers
    - test extraneous messages and ensure they are handled
- 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
    - getClients()
        - check that the clients list is returned properly (JUnit)
    - addClient(Client c)
        - ensure client is added to the clients list (JUnit)
        - test case when 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)
    - sendMessageToAll(String message)
        - ensure runs properly (JUnit) (we test for actual sending later)
    - drawLine(x1, y1, x2, y2, width, r, g, b)
        - observe that a line is drawn on the GUI with the correct specifications
    - resetHistory()
        - ensure history list is empty after calling this
        - visually check that bmp file is up to date (same image as GUI)
    - checkHistory()
        - test when size of history list is both above and below chosen threshold
    - addLine(String message)
        - check that line drawn on GUI is correct
        - check that line is added to history list (JUnit)
    - sendBMP(File bmp, Client c)
        - ensure runs properly (JUnit)
    - sendHistory(List<String> history, Client c)
        - ensure runs properly (JUnit)
    - loadWhiteboard(Client c)
        - ensure runs properly (JUnit)
- Client
    - sendMessage(String message)
        - ensure runs properly (JUnit)

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

- 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