6.005 Project 2 Revised Design

I. Datatype Design:

- **Model:** How we will represent each whiteboard in memory
  - **Class:** Point
    - **Description:** Represents a pixel in a window, each pixel in a whiteboard will be stored in memory as a Point object.
    - **Invariants:**
      - Mutable
      - Threadsafe
    - **Fields:**
      - *final int x* -- the x-coordinate of the point
      - *final int y* -- the y-coordinate of the point
      - *Color color* -- the color of the point
    - **Constructors:**
      - *public Point (final int x, final int y, final Color color)* -- creates a Point object
    - **Methods:**
      - *public int getX()* -- returns the x-coordinate of the Point object
      - *public int getY()* -- returns the y-coordinate of the Point object
      - *public synchronized Color getColor()* -- returns the color of the Point object
      - *public synchronized void setColor(Color new Color)* -- changes the color of the Point object
      - *public synchronized String toString()* -- returns the String representation of the Point object
  - **Class:** Whiteboard
    - **Description:** Represents a collaborative whiteboard, basically consisting of a 2D array of pixels or Point objects
    - **Invariants:**
      - Mutable
      - Threadsafe
    - **Fields:**
      - *final String name* -- the name of the whiteboard
      - *final int height* -- the height of the whiteboard in pixels
      - *final int width* -- the width of the whiteboard in pixels
      - *final Point[][] pixels* -- a width x height array of Point object, each Point object representing a pixel in the whiteboard
        - The Point object at pixels[x][y] represents the pixel at (x, y) where (0, 0) is located in the top left hand corner of the whiteboard and (width -1, height - 1) is located in the bottom right hand corner of the whiteboard.
      - *final ArrayList<String> usernames* -- the usernames of all of the

client that have this whiteboard open

        - **Constructors:**
            - *public Whiteboard (final String name, final int width, final int height, final Color startingColor)* -- creates an empty Whiteboard object.
        - **Methods:**
            - *public syncrhonized String getUsernames()* -- returns a string containing the usernames of all of the clients that currently have this whiteboard open, separated by single spaces
            - *public synchronized String addUsername()* -- adds a username to this whiteboard, signifying that a client has connected to this whiteboard
            - *public synchronized String removeUsername()* -- removes a username from this whiteboard, signifying that a client has disconnected from this whiteboard
            - *public String getName()* -- returns the name of the whiteboard object
            - *public getHeight()* -- returns the height of the whiteboard in pixels
            - *public getWidth()* -- returns the width of the whiteboard in pixels
            - *public setColor(final int x, final int y, final Color newColor)* -- changes the color of a pixel in the whiteboard
            - *public synchronized String addLine(final Color color, final int x1, final int y1, final int x2, final int y2, final int thickness)* -- adds a line segment to this Whiteboard object and returns a string representation of this line
            - *protected ArrayList<Point> getPointsOnLine(final Color color, final int x1, final int y1, final int x2, final int y2)* -- determines all of the points between the starting and ending points of a line segment
            - *protected String colorPoints* -- colors all of the point specified with the specified color
            - *public synchronized String toString()* -- returns the string representation of this Whiteboard

- **Server-Client:** How we will represent clients and servers
    - **Class:** WhiteboardServer
        - **Description:** Represents the server of the Whiteboard constructors
        - **Constructors:**
            - *public WhiteboardServer(final int port)* -- creates a WhiteboardServer that listens for conenctions on port
        - **Fields:**
            - *private final ServerSocket serverSocket* -- the socket to which the server is listening for client connections
            - *private final ArrayList<Whiteboard> whiteboards* -- a threadsafe encapsulation of the whiteboards that have been created during a

single session
- *private final ArrayList<String> names* -- a threadsafe encapsulation of the names of the whiteboards thta have been created during a single session
  - The name of the ith Whiteboard object in whiteboards is the ith String in names
- *private final HashMap<String, Socket> clients* -- the keys are the usernames of the clients that are connected to this WhiteboardServer and the values are the sockets via which each client is connected to the server

- **Methods:**
  - *public void serve()* -- runs the server, listening for client connections and handling them.
  - *private void handleConnection()* -- handles a single client connection. Returns when a client disconnects.
  - *private synchronized Whiteboard handleRequest(String name)* -- handles a client request for a whiteboard

- **GUI:** How we will show the whiteboards
  - **Class:** Canvas
    - **Description:** This builds the mutable space for the client to write on and edit as they please. This was already supplied, but a few methods have been edited and added to.
    - Methods:
      - drawLineSegment()
        - a change was made to handle the bar that controls the width of the cursor's line.
        - a change was made to handle multiple colors.
      - setColor()
        - This was added to set the color of the line when the button on the GUI was pressed.
      - Main()
        - This was edited to create an instance of WhiteBoardGUI
        - The Create method in the GUI is called in the main of Canvas to create the window of the GUI.
  - **Class:** WhiteboardGUI
    - **Description:** This builds the actual window display and the buttons that is contains. It pushes all of the computation to the canvas class to repaint the canvas.
    - Methods:
      - *public void WhiteBoardGUI()*
        - Creates all the action listeners for the various buttons as well as populates the contentPane with the toolBar and

canvas.

- *public voidToolBarButtons()*
    - This adds all of the buttons to the toolbar and builds the layout of the toolBar.
- *public void CreateGUI()*
    - This sets the size of the window, populates the window with the content pane and adds a couple functionalities to the window.

II. Protocol:

create [name] -

If [name] contains any spaces, this method returns an output message of the form "Whiteboard names cannot contain any spaces."
If a whiteboard whose name is [name] is saved on the server (i.e. has already been created), this method returns an output message of the form "A whiteboard with that name has already created. Please choose another name."
Otherwise, if a whiteboard whose name is [name] is not saved on the server (i.e. has either never been created or was deleted) and [name] does not contain any spaces, this method creates a whiteboard whose name is [name] and returns the name of the whiteboard this method also notifies all other clients of this change with a message of the form all usernames [whiteboard 1] [whiteboard 2] [whiteboard 3] ... [whiteboard n] where [whiteboard 1] [whiteboard 2] [whiteboard 3] ... [whiteboard n] are the names of all of the whiteboards on the server

open [username] [name] -

[username] must represent the client that is connecting through the currently handled port.
If a whiteboard whose name is [name] is saved on the server, this method returns a string representation of the whiteboard to the client and adds the client to the specified whiteboard this method also notifies all clients currently connected to the specified whiteboard of the new client
Otherwise, then returns an output message of the form "A whiteboard with that name does not exist."

logout [username] -

[username] must represent the client that is connecting through the currently handled port.
This method returns an empty output message and removes the client from all whiteboards that it is connected to and notifies all other clients both of the fact that they have disconnected from the whole server and that they have disconnected from all of the whiteboards

username [username] -

If a client whose username is [username] is connected to the server, this method returns "That username is already being used."
If [username] contains a space, this method returns "There should be no spaces in the username."
Otherwise, this method creates an entry in the HashMap corresponding to this username and

client and returns the requested username

getusernames [name]
[name] must be a whiteboard on the server
This method returns the usernames of all of the whiteboards on the server

close [username] [name] -
[username] must a client on the server that is connected to whiteboard
[name] must be a whiteboard on the server
This method removes the client whose [username] from whiteboard whose name is [name]

draw [name] [x1] [x2] [y1] [y2] [red] [green] [blue] [thickness] -
[name] must be a whiteboard on the server
This method requests a line to be draw on the whiteboard named [name] from ([x1], [y1]) to ([x2], [y2]) with color [red] [green] [blue] and thickness [thickness]
Note that ([x1], [y1]) and ([x2], [y2]) must be points on the whiteboard named [name]
This method returns an empty string.
This method also sends out a message to all of the clients connected to the whiteboard [name] that this line has been added to the whiteboard

III. Concurrency Strategy:
The classes Point and Whiteboard should be both threadsafe (as indicated in the section Datatype Design). The Whiteboard object should be synchronized on Point objects so that each multiple servers can change different parts of the same Whiteboard object at the same time, but they cannot change the same part of the same Whiteboard object at the same time.

There will be a main thread for the server and a different thread handling each of the clients. Each client will also have a GUI thread handling the whiteboard display window.

IV. Testing Strategy:

- 3 Levels of Testing
    - Unit Testing
    - Testing via Isolation
    - Integrated Process
- Unit Testing
    - Point datatype methods
    - Whiteboard datatype methods
    - WhiteBoardGUI
        - manual testing
            - how can we break the program/how will it be used?
        - action listeners
            - JUnit test if possible, if not, manual test
        - testing Canvas class and its methods

    - ClientLogIn (GUI)
        - manual testing
            - how can we break the program/how will it be used?
        - action listeners
            - JUnit test if possible, if not, manual test
    - WhiteboardServer methods
- Testing via Isolation
    - WhiteboardServer
        - Hardcoded messages being passed in and making sure the server reacts in the correct manner
    - WhiteBoardGUI - client
        - Hardcoded messages being passed in and making sure the GUI is reacting in the correct manner
    - ClientLogIn (GUI)
        - Hardcoded messages being passed in and making sure the GUI is reacting in the correct manner
- One Test To Rule Them All (Integrated Test)
    - Logging in
        - Signing in as a user
            - Either creating or retrieving a board
    - Editing a whiteboard
        - Multiple users editing whiteboard
    - Sending changes to server and making the server sends back the appropriate messages to client
        - Client’s GUI responding in the correct manner after receiving the messages