\*\*\*\*\*\*\*\* Canvas.java\*\*\*\*\*\*\*\*\*

package client;

import java.awt.AlphaComposite;

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Image;

import java.awt.Point;

import java.awt.RenderingHints;

import java.awt.Shape;

import java.awt.image.BufferedImage;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.awt.geom.Rectangle2D;

import java.io.File;

import java.io.IOException;

import java.util.EventListener;

import javax.imageio.ImageIO;

import javax.swing.\*;

/\*\*

\* Canvas represents a drawing surface that allows the user to draw

\* on it freehand, with the mouse.

\*/

public class Canvas extends JPanel {

// image where the user's drawing is stored

private static final long serialVersionUID = 2L;

private final Client client;

private EventListener currentListener;

public Canvas(Client client) {

this.client = client;

}

@Override

public void paintComponent(Graphics g) {

// If this is the first time paintComponent() is being called,

// make our drawing buffer.

if (client.getDrawingBuffer() == null) {

makeDrawingBuffer();

}

// Copy the drawing buffer to the screen.

g.drawImage(client.getDrawingBuffer(), 0, 0, null);

}

/\*\*

\* Make the drawing buffer and draw some starting content for it.

\*/

protected void makeDrawingBuffer() {

client.setDrawingBuffer(new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE\_INT\_RGB));

fillWithWhite();

}

/\*\*

\* Make the drawing buffer entirely white.

\*/

protected void fillWithWhite() {

final Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

g.setColor(Color.WHITE);

g.fillRect(0, 0, getWidth(), getHeight());

this.repaint();

}

/\*\*

\* Draw a selected image from file

\*/

public void fillWithImage(String ImagePath) {

final Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

final BufferedImage img;

ImageIcon MyImage = new ImageIcon(ImagePath);

try {

img = ImageIO.read(new File(ImagePath));

g.drawImage(img, 0, 0, this);

} catch (IOException ex) {

ex.printStackTrace();

}

this.repaint();

}

/\*\*

\* Draw a selected image from file and put it on server

\*/

protected void fillWithImageAndCall(String ImagePath) {

fillWithImage(ImagePath);

try {

client.makeDrawRequest("fillWithImage "+ImagePath);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Draw a line between and send it to every client

\*/

protected void drawLineSegmentAndCall(int x1, int y1, int x2, int y2, int color, float width) {

drawLineSegment(x1, y1, x2, y2, color, width);

try {

client.makeDrawRequest("drawLineSegment "+x1+" "+y1+" "+x2+" "+y2+" "+(color+16777216)+" "+width);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Draw a line between two points (x1, y1) and (x2, y2), specified in

\* pixels relative to the upper-left corner of the drawing buffer.

\*/

public void drawLineSegment(int x1, int y1, int x2, int y2, int color, float width) {

Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

Color colorObject = new Color(color);

g.setColor(colorObject);

new BasicStroke( width,// Width

BasicStroke.CAP\_ROUND, // End cap

BasicStroke.JOIN\_ROUND, // Join style

10.0f, // limit

null, // Dash pattern

0.0f);

g.setStroke(new BasicStroke( width,// Width

BasicStroke.CAP\_ROUND, // End cap

BasicStroke.JOIN\_ROUND, // Join style

10.0f, // limit

null, // Dash pattern

0.0f));

g.drawLine(x1, y1, x2, y2);

this.repaint();

}

/\*\*

\* Save the canvas as .png on file

\*/

public void saveMethod() {

BufferedImage bi = new BufferedImage(800, 600, BufferedImage.TYPE\_INT\_ARGB);

Graphics g = bi.createGraphics();

this.paint(g);

g.dispose();

JFileChooser fc = new JFileChooser();

int returnVal = fc.showSaveDialog(this);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

File file = fc.getSelectedFile();

try{

ImageIO.write(bi,"png",file);

}

catch (Exception e) {

e.printStackTrace();

}

}

}

//tring to make the square show where it is drawn

public void newDrawSquare(Point startDrag, Point endDrag) {

Graphics2D g2 = (Graphics2D) client.getDrawingBuffer().getGraphics();

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

Color[] colors = { Color.YELLOW, Color.MAGENTA, Color.CYAN , Color.RED, Color.BLUE, Color.PINK};

int colorIndex = 0;

g2.setStroke(new BasicStroke(2));

g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 0.50f));

if (startDrag != null && endDrag != null) {

g2.setPaint(Color.LIGHT\_GRAY);

Shape r = makeRectangle(startDrag.x, startDrag.y, endDrag.x, endDrag.y);

g2.draw(r);

System.out.println("am ajuns aici"+" "+ startDrag.x + " " + startDrag.y + " " + endDrag.x+ " " +endDrag.y);

}

this.setVisible(true);

this.repaint();

}

private Rectangle2D.Float makeRectangle(int x1, int y1, int x2, int y2) {

return new Rectangle2D.Float(Math.min(x1, x2), Math.min(y1, y2), Math.abs(x1 - x2), Math.abs(y1 - y2));

}

/\*\*

\* Draw a rectagle

\*/

public void drawSquare(int x, int y, int x2, int y2, int color, float width) {

Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

Color colorObject = new Color(color);

g.setColor(colorObject);

g.setStroke(new BasicStroke(width));

int px = Math.min(x,x2);

int py = Math.min(y,y2);

int pw=Math.abs(x-x2);

int ph=Math.abs(y-y2);

g.drawRect(px, py, pw, ph);

this.repaint();

}

/\*\*

\* Draw a rectagle and call

\*/

protected void drawSquareAndCall(int x1, int y1, int x2, int y2, int color, float width) {

drawSquare(x1, y1, x2, y2, color, width);

try {

client.makeDrawRequest("drawSquare "+x1+" "+y1+" "+x2+" "+y2+" "+(color+16777216)+" "+width);

} catch (IOException e) {

e.printStackTrace();

}

}

protected void drawTextAndCall(String textFont, int color, int width, int idfk, int x, int y) {

drawText(textFont, color,width, 0, x, y);

try {

client.makeDrawRequest("drawText "+textFont+" "+(color+16777216)+" "+width+" "+idfk+" "+x+" "+y);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Edit where the text will be draw

\*/

public void drawText(String textFont,int color, int width, int idfk, int x, int y) {

String text= "";

text = JOptionPane.showInputDialog(this, "Text to add", "Text");

String textBuffer = "";

Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

Font font = new Font(textFont, Font.PLAIN, width);

FontMetrics metrics = g.getFontMetrics(font);

int i = 0;

int j = 0;

int k = 0;

while(i < text.length()) {

if(i == 0) {

textBuffer += text.charAt(i);

i++;

}

textBuffer += text.charAt(i);

int maxDim = 550 - x - metrics.stringWidth(" ");

if(metrics.stringWidth(text.substring(j, i)) >= maxDim) {

Color colorObject = new Color(color);

drawOnlyText(textBuffer,textFont, color, width, x, y, k);

drawOnlyTextAndCall(textBuffer,textFont, color, width, x, y, k);

System.out.println("draw");

j = i + 1;

k+=metrics.getHeight();

textBuffer = "";

}

else if(metrics.stringWidth( text.substring(j, text.length()-1)) < (550 - x) ) {

textBuffer = text.substring(j, text.length());

Color colorObject = new Color(color);

drawOnlyText(textBuffer,textFont, color, width, x, y, k);

drawOnlyTextAndCall(textBuffer,textFont, color, width, x, y, k);

System.out.println("draw");

break;

}

i++;

}

}

/\*\*

\* Acttually draw text and call

\*/

public void drawOnlyTextAndCall(String textBuffer, String textFont, int color, int width, int x, int y, int k) {

try {

client.makeDrawRequest("drawOnlyText "+textBuffer+" "+textFont+" "+(color+16777216)+" "+width+" "+x+" "+y+" "+k);

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Acttually draw text

\*/

public void drawOnlyText(String textBuffer, String textFont, int color, int width, int x, int y, int k) {

Graphics2D g = (Graphics2D) client.getDrawingBuffer().getGraphics();

Color colorObject = new Color(color);

Font font = new Font(textFont, Font.PLAIN, width);

g.setColor(colorObject);

g.setFont(font);

g.drawString(textBuffer, x, (y + k));

this.repaint();

}

/\*\*

\* Updates the label showing the current username and the current board name

\*/

public void updateCurrentUserBoard() {

String user = client.getUsername();

String board = client.getCurrentBoardName();

client.getClientGUI().setCurrentUserBoard(new JLabel("Hi, " + user + ". This board is: " + board));

}

/\*\*

\* Add the mouse listener that supports the user's freehand drawing.

\*/

public void addDrawingController(EventListener listener) {

if (currentListener != null) {

removeMouseListener((MouseListener) currentListener);

removeMouseMotionListener((MouseMotionListener) currentListener);

}

currentListener = listener;

addMouseListener((MouseListener) currentListener);

addMouseMotionListener((MouseMotionListener) currentListener);

}

/\*\*

\* Resets the drawing buffer to be blank and calls switch canvas on the client

\* @param board

\*/

public void switchBoard(String board) {

fillWithWhite();

client.switchBoard(board);

}

public EventListener getCurrentListener() {

return currentListener;

}

}

\*\*\*\*\*\*\*\*\*\*\*\* Client.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package client;

import java.awt.Color;

import java.awt.image.BufferedImage;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.net.UnknownHostException;

import java.util.Arrays;

import java.util.Hashtable;

import javax.swing.SwingUtilities;

import command.Command;

/\*

\* Client class that controls what one user does on the Board(ClientGUI)

\*/

public class Client {

private String username;

private String currentBoardName;

//the color the user is currently drawing in

private Color currentColor = Color.BLACK;

//the width of the brush the user is currently drawing with

private float currentWidth = 10;

private BufferedImage drawingBuffer;

// used for server-client communications:

// All data updated by server requests must also have a tracker as to whether it

// has been updated. All server responses are handled in a seperate thread so as

// to enable real time updates. This means that each request for an update must

// wait on these tracker variables to confirm that the server has responded with

// the updated information

private String[] boards = {};

private final long timeoutLength = 500L;

private Tracker boardsUpdated = new Tracker(false);

private Hashtable<String, Tracker> newBoardMade = new Hashtable<String, Tracker>();

private Hashtable<String, Tracker> newBoardSuccessful = new Hashtable<String, Tracker>();

private Tracker userCheckMade = new Tracker(false);

private Tracker usersUpdated = new Tracker(false);

private String[] users = {};

private Tracker exitComplete = new Tracker(false);

//booleans for different modes of painting

private boolean isErasing;

private boolean squareOn;

private boolean eraserOn;

private boolean brushOn;

private boolean straightOn;

private boolean dottedOn;

private boolean dashedOn;

private boolean arialOn = true;

private boolean comicOn;

private boolean textOn;

//the socket with which the user connects to the client

private Socket socket;

BufferedReader in;

PrintWriter out;

ClientReceiveProtocol receiveProtocol;

Thread receiveThread;

private ClientGUI clientGUI;

/\*\*

\* Starts a whiteboard client connected to host on the given port.

\*

\* @param host

\* @param port

\* @throws UnknownHostException

\* @throws IOException

\*/

public Client(String host, int port) throws UnknownHostException, IOException {

socket = new Socket(host, port);

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

out = new PrintWriter(socket.getOutputStream(), true);

receiveProtocol = new ClientReceiveProtocol(in, this);

receiveThread = new Thread(receiveProtocol);

receiveThread.start();

addShutdownHook();

}

/\*\*

\* Starts the client's GUI

\*/

public void startGUI() {

clientGUI = new ClientGUI(this);

}

/\*\*

\* Checks with the server to make sure the username hasn't already been taken and if it hasn't, create the user

\* @param username: the user's choice of username

\* @return: true if username creation is successful, false if not

\*/

public boolean createUser(String username, String boardName) throws Exception {

// Make request and wait for a response

userCheckMade.setValue(false);

makeRequest("checkAndAddUser "+username+" "+boardName).join();

timeout(userCheckMade, timeoutLength, "check and add user");

return (this.username != null && currentBoardName != null);

}

/\*\*

\* Parses and stores response from server after newUser request is made

\* @param response: String response from the server

\* @throws Exception

\*/

public void parseNewUserFromServerResponse(String response) throws Exception {

String[] elements = response.split(" ");

if(elements[0]!="check"&& elements.length!=4) {

throw new Exception("Server returned unexpected result: " + response);

}

boolean created = Boolean.valueOf(elements[3]);

if (created) {

this.username = elements[1];

this.currentBoardName = elements[2];

}

userCheckMade.setValue(true);

}

/\*\*

\* Switches the current board to the board with the given name

\* server switch command

\* Updates the current users of the canvas

\* @param newBoardName: the name of the new board

\*/

public void switchBoard(String newBoardName) {

try {

makeRequest("switch "+username+" "+currentBoardName+" "+newBoardName);

currentBoardName = newBoardName;

getCanvas().updateCurrentUserBoard();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Makes request to draw on the server

\* @param command

\* @throws IOException

\*/

public void makeDrawRequest(String command) throws IOException {

makeRequest("draw "+currentBoardName+" "+command);

}

/\*\*

\* Check that the boardName and currentBoardName are the same and then perform the command on the canvas

\* @param boardName: the board this command is for

\* @param command: the command to perform on the canvas

\*/

public void commandCanvas(String boardName, Command command) {

if (command.checkBoardName(boardName)) {

command.invokeCommand(getCanvas());

}

}

/\*\*

\* invokes command on canvas

\* @param command: command to be applied to canvas

\*/

public void applyCommand(Command command) {

command.invokeCommand(getCanvas());

}

/\*\*

\* Checks that the board name hasn't already been taken and if it hasn't,

\* creates a new board on the server and names it with the given name

\*

\* @param newBoardName

\* the name to name the new board with

\* @return true if the board creation is successful, false if not

\*/

public boolean newBoard(String newBoardName) throws Exception {

if (newBoardMade.containsKey(newBoardName)) return false;

// make request and wait for response

newBoardMade.put(newBoardName, new Tracker(false));

newBoardSuccessful.put(newBoardName, new Tracker(true));

makeRequest("newBoard "+newBoardName).join();

timeout(newBoardMade.get(newBoardName), timeoutLength, "new board");

// check if we got a successful response

boolean successful = newBoardSuccessful.get(newBoardName).getValue();

newBoardMade.remove(newBoardName);

newBoardSuccessful.remove(newBoardName);

return successful;

}

/\*\*

\* Parses and stores response from server after new board request is made

\* @param response: response from server

\* @throws Exception

\*/

public void parseNewBoardFromServerResponse(String response) throws Exception {

if(!response.contains("newBoard")) {

throw new Exception("Server returned unexpected result: " + response);

}

String[] elements = response.split(" ");

String boardName = elements[1];

boolean successful = Boolean.valueOf(elements[2]);

Tracker nbs = newBoardSuccessful.get(boardName);

if (!(nbs == null)) { nbs.setValue(successful); }

Tracker nbm = newBoardMade.get(boardName);

if (!(nbm == null)) { nbm.setValue(true); }

}

/\*\*

\* Gets the users for the current board from the server and sets them

\*/

public String[] getUsers() throws Exception {

// Make request for users and wait for response

usersUpdated.setValue(false);

makeRequest("users "+currentBoardName);

timeout(usersUpdated, timeoutLength, "Updateing Users");

return users;

}

/\*\*

\* Sets users

\* @param newUsers

\*/

public void setUsers(String[] newUsers) {

users = newUsers;

usersUpdated.setValue(true);

}

/\*\*

\* Parses and stores response from server after users request is made

\* @param response: response from server

\* @return: String[] of users on baord

\* @throws Exception

\*/

public String[] parseUsersFromServerResponse(String response) throws Exception {

String[] elements = response.split(" ");

if(!elements[0].equals("users")) {

throw new Exception("Server returned unexpected result: " + response);

}

return Arrays.copyOfRange(elements, 2, elements.length);

}

/\*\*

\* Gets new boards from the server. Makes an update request and returns the results.

\* @return: String[] of board names currently stored on the server

\* @throws Exception

\*/

public String[] getBoards() throws Exception {

boardsUpdated.setValue(false);

// make request for board update and wait for it to finish

makeRequest("boards").join();

timeout(boardsUpdated, timeoutLength, "boards update");

// boards by now will have either been updated, or if it times out

// then it will return what it last had

return this.boards;

}

/\*\*

\* Sets current baords on server to newBoards

\* @param newBoards: String[] of boards to set as newBoards

\*/

public void setBoards(String[] newBoards) {

boards = newBoards;

boardsUpdated.setValue(true);

}

/\*\*

\*

\* Parses and stores response from server after boards request is made

\* @param response: response from server

\* @return: String[] of boards on server

\* @throws Exception

\*/

public String[] parseBoardsFromServerResponse(String response) throws Exception {

if(!response.contains("boards")) {

throw new Exception("Server returned unexpected result: " + response);

}

String[] boardsListStrings = response.split(" ");

return Arrays.copyOfRange(boardsListStrings, 1, boardsListStrings.length);

}

/\*\*

\* Timeout function. Waits on Tracker passed in to change to true, or returns after

\* a certain timeout length and prints a timeout error. Returns true if it does

\* time out, false otherwise.

\*

\* @param variable: Tracker object to watch for a change to true

\* @param timeoutLength: Time to wait before timing out

\* @param timeoutMessage: Message to append to error log in event of time out

\* @return: true if it times out, false if variable becomes true before then

\*/

public boolean timeout(Tracker variable, long timeoutLength, String timeoutMessage) {

long startTime = System.currentTimeMillis();

// Keep going until the variable becomes true or we timeout

while(!variable.getValue()) {

long currentTime = System.currentTimeMillis();

if(currentTime >= startTime + timeoutLength) {

System.err.println("Timeout on: " + timeoutMessage);

return true;

}

}

return false;

}

/\*\*

\* Makes request passed in to server

\* @param request: String of request you want to send

\* @return: Thread with request running in it

\* @throws IOException

\*/

public Thread makeRequest(String request) throws IOException {

Thread requestThread = new Thread(new ClientSendProtocol(out, request));

requestThread.start();

return requestThread;

}

/\*\*

\* Checks if currentBoard is equal to boardName passed in

\* @param boardName: boardName to check

\* @return: true if boardName is equal to currentBoardName stored in Client

\*/

public boolean checkForCorrectBoard(String boardName) {

return boardName.equals(currentBoardName);

}

// Basic getters and setters:

/\*\*

\* Get GUI for client

\* @return

\*/

public ClientGUI getClientGUI() {

return clientGUI;

}

/\*\*

\* Gets the current color to use for drawing a line segment on the canvas

\* @return the currentColor being used to draw

\*/

public Color getCurrentColor() {

return currentColor;

}

/\*\*

\* Gets the current width to use for drawing a line segment on the canvas

\* @return the currentWidth being used to draw

\*/

public float getCurrentWidth() {

return currentWidth;

}

/\*\*

\* Sets the newWidth, probably based off of a slider movement on the canvas

\* @param newWidth: the new width of the stroke

\*/

public void setCurrentWidth(float newWidth) {

currentWidth = newWidth;

}

/\*\*

\* Sets the newColor

\* @param newWidth: the new color of the stroke

\*/

public void setCurrentColor(Color newColor) {

currentColor = newColor;

}

/\*\*

\* Sets username

\* @param username: new username

\*/

public void setUsername(String username) {

this.username = username;

}

/\*\*

\* Sets current board name

\* @param currentBoardName: new name to store as current baord name

\*/

public void setCurrentBoardName(String currentBoardName) {

this.currentBoardName = currentBoardName;

}

/\*\*

\* Gets current board name

\* @return: current board name

\*/

public String getCurrentBoardName() {

return currentBoardName;

}

/\*\*

\* Returns canvas

\* @return

\*/

public Canvas getCanvas() {

return clientGUI.getCanvas();

}

/\*\*

\* Returns username

\* @return

\*/

public String getUsername() {

return username;

}

/\*\*

\* Getter for drawingBuffer

\* @return

\*/

public BufferedImage getDrawingBuffer() {

return drawingBuffer;

}

public void setFontArial(boolean on) {

this.arialOn = on;

}

public boolean getFontArial() {

return this.arialOn;

}

public void setFontComic(boolean on) {

this.comicOn = on;

}

public boolean getFontComic() {

return this.comicOn;

}

public void setText(boolean on) {

this.textOn = on;

}

public boolean getText() {

return this.textOn;

}

public void setStraight(boolean on) {

this.straightOn = on;

}

public boolean getStraight() {

return this.straightOn;

}

public void setDotted(boolean on) {

this.dottedOn = on;

}

public boolean getDotted() {

return this.dottedOn;

}

public void setDashed(boolean on) {

this.dashedOn = on;

}

public boolean getDashed() {

return this.dashedOn;

}

/\*\*

\* Setter for SquareOn

\* @param newImage

\*/

public void setSquareOn(boolean on) {

this.squareOn = on;

}

/\*\*

\* Getter for SquareOn

\* @return: true if isErasing is set to true

\*/

public boolean getSquareOn() {

return this.squareOn;

}

/\*\*

\* Setter for SquareOn

\* @param newImage

\*/

public void setEraserOn(boolean on) {

this.eraserOn = on;

}

/\*\*

\* Getter for BrushOn

\* @return: true if isErasing is set to true

\*/

public boolean getEraserOn() {

return this.eraserOn;

}

public void setBrushOn(boolean on) {

this.brushOn = on;

}

/\*\*

\* Getter for SquareOn

\* @return: true if isErasing is set to true

\*/

public boolean getBrushOn() {

return this.brushOn;

}

/\*\*

\* Setter for drawingBuffer

\* @param newImage

\*/

public void setDrawingBuffer(BufferedImage newImage) {

drawingBuffer = newImage;

}

/\*\*

\* Set isErasing variable to newIsErasing

\* @param newIsErasing: boolean to set isErasing to

\*/

public void setIsErasing(boolean newIsErasing) {

isErasing = newIsErasing;

}

/\*\*

\* Getter for isErasing

\* @return: true if isErasing is set to true

\*/

public boolean isErasing() {

return isErasing;

}

/\*\*

\* Confirms exit of client from server

\*/

public void completeExit() {

exitComplete.setValue(true);

}

public void kill() {

try {

// kill receiving thread and wait for it to close out

if (username!= null) {

try {

exitComplete.setValue(false);

makeRequest("exit "+username).join();

timeout(exitComplete, timeoutLength, "Exiting");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

receiveProtocol.kill();

if(!socket.isClosed()) {

socket.shutdownInput();

socket.shutdownOutput();

socket.close();

}

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Adds commands to shutdown sequence in order to close Client gracefully

\*/

public void addShutdownHook() {

// close socket on exit

Runtime.getRuntime().addShutdownHook(new Thread()

{

@Override

public void run()

{

kill();

}

});

}

/\*\*

\* For testing purposes. Gets the ClientReceiveProtocol

\* @param args

\*/

public ClientReceiveProtocol getClientReceiveProtocol() {

return receiveProtocol;

}

/\*\*

\* Testing purposes for new board method

\* @return

\*/

public Hashtable<String, Tracker> getBoardSuccessful() {

return newBoardSuccessful;

}

/\*\*

\* Get exitComplete

\*/

public Tracker getExitComplete() {

return exitComplete;

}

/\*

\* Main program. Make a window containing a Canvas.

\*/

public static void main(String[] args) {

// set up the UI (on the event-handling thread)

SwingUtilities.invokeLater(new Runnable() {

public void run() {

try {

Client client = new Client("localhost", 4444);

client.startGUI();

} catch (UnknownHostException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

});

}

}

\*\*\*\*\*\*\*\*\*\*\*\* ClintGUI.java \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package client;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Container;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.Point;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.File;

import javax.swing.BorderFactory;

import javax.swing.Box;

import javax.swing.BoxLayout;

import javax.swing.DefaultListModel;

import javax.swing.GroupLayout;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JColorChooser;

import javax.swing.JDialog;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JPopupMenu;

import javax.swing.JScrollPane;

import javax.swing.JSlider;

import javax.swing.JTextField;

import javax.swing.JToggleButton;

import javax.swing.ListSelectionModel;

import javax.swing.SwingConstants;

import javax.swing.SwingWorker;

import javax.swing.border.TitledBorder;

import javax.swing.GroupLayout.Group;

import javax.swing.GroupLayout.ParallelGroup;

import javax.swing.GroupLayout.SequentialGroup;

import javax.swing.GroupLayout.Alignment;

import javax.swing.colorchooser.AbstractColorChooserPanel;

import javax.swing.colorchooser.ColorSelectionModel;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

import javax.swing.event.MenuEvent;

import javax.swing.event.MenuListener;

import javax.swing.filechooser.FileNameExtensionFilter;

import static javax.swing.GroupLayout.Alignment.BASELINE;

import static javax.swing.GroupLayout.Alignment.LEADING;

public class ClientGUI extends JFrame {

private static final long serialVersionUID = 1L;

private final Client client;

private final int WIDTH = 800;

private final int HEIGHT = 600;

private JToggleButton imageToggle;

private JToggleButton text;

private JToggleButton cropToggle;

private JToggleButton brush;

private JToggleButton ereser;

private JToggleButton square;

private JToggleButton triangle;

private JToggleButton circle;

private JButton strokeButton;

// Start Dialog GUI objects

private JDialog dialog;

private DefaultListModel<String> boardListModel;

private JLabel newBoardLabel;

private JTextField newBoard;

private JList<String> boardList;

private Container dialogContainer;

private GroupLayout layout;

private JTextField usernameTextField;

private JLabel usernameLabel;

private JScrollPane boardListScroller;

private JButton newBoardButton;

private JButton startButton;

private JLabel currentUserBoard;

private JPanel sidebar;

private Canvas canvas;

/\*\*

\* Makes the view for the client (the model)

\* @param client: the client this GUI works for

\*/

public ClientGUI(Client client) {

this.client = client;

startDialog();

}

/\*\*

\* Creates start dialog which handles username and initial board

\*/

private void startDialog() {

dialog = new JDialog();

dialog.setTitle("Welcome to Whiteboard");

dialog.setResizable(false);

dialog.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setDialogLayout();

setDialogActionListeners();

}

/\*\*

\* Sets layout for start dialog

\*/

public void setDialogLayout() {

dialogContainer = new Container();

layout = new GroupLayout(dialogContainer);

layout.setAutoCreateGaps(true);

layout.setAutoCreateContainerGaps(true);

dialogContainer.setLayout(layout);

ParallelGroup hGroup = layout.createParallelGroup(GroupLayout.Alignment.LEADING);

SequentialGroup hUsername = layout.createSequentialGroup();

usernameTextField = new JTextField(10);

usernameTextField.setName("username");

usernameLabel = new JLabel("Username:");

hUsername.addComponent(usernameLabel).addComponent(usernameTextField);

boardListModel = new DefaultListModel<String>();

// Get boards from server and add to data model

try {

String[] boards = client.getBoards();

for (int i=0; i<boards.length;i++) {

boardListModel.addElement(boards[i]);

}

} catch (Exception e1) {

e1.printStackTrace();

}

boardList = new JList<String>(boardListModel); //data has type Object[]

boardList.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

boardList.setLayoutOrientation(JList.VERTICAL);

boardList.setVisibleRowCount(-1);

boardListScroller = new JScrollPane(boardList);

boardListScroller.setPreferredSize(new Dimension(100, 150));

SequentialGroup hNewBoard = layout.createSequentialGroup();

newBoardLabel = new JLabel("New Board:");

newBoard = new JTextField(10);

newBoard.setName("newBoard");

newBoardButton = new JButton("Add Board");

hNewBoard.addComponent(newBoardLabel).addComponent(newBoard).addComponent(newBoardButton);

startButton = new JButton("Start");

hGroup.addGroup(hUsername).addComponent(boardListScroller).addGroup(hNewBoard).addComponent(startButton);

ParallelGroup vGroup = layout.createParallelGroup(GroupLayout.Alignment.LEADING);

SequentialGroup vAll = layout.createSequentialGroup();

ParallelGroup v1 = layout.createParallelGroup(GroupLayout.Alignment.BASELINE);

v1.addComponent(usernameLabel).addComponent(usernameTextField);

ParallelGroup v2 = layout.createParallelGroup(GroupLayout.Alignment.BASELINE);

v2.addComponent(newBoardLabel).addComponent(newBoard).addComponent(newBoardButton);

vAll.addGroup(v1).addComponent(boardListScroller).addGroup(v2).addComponent(startButton);

vGroup.addGroup(vAll);

layout.setHorizontalGroup(hGroup);

layout.setVerticalGroup(vGroup);

dialog.setContentPane(dialogContainer);

dialog.pack();

dialog.setVisible(true);

}

/\*\*

\* Adds action listeners to start dialog

\*/

private void setDialogActionListeners() {

//handles when the initial dialog is closed; kills the client

dialog.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

client.kill();

}

});

//handles what happens when the user presses start

//enters the user is possible and sets up the canvas

startButton.addActionListener(new ActionListener() {

public synchronized void actionPerformed(ActionEvent e) {

//if the username is empty

if (usernameTextField.getText().equals("")) {

JOptionPane.showMessageDialog(dialog, "Please enter a username.", "Try again", JOptionPane.ERROR\_MESSAGE);

}

//if no board is selected

else if (boardList.isSelectionEmpty()) {

JOptionPane.showMessageDialog(dialog, "Please select a board.", "Try again", JOptionPane.ERROR\_MESSAGE);

} else

try {

//if the user has entered successfully

if (client.createUser(usernameTextField.getText(), boardList.getSelectedValue())) {

dialog.dispose();

setupCanvas();

client.makeRequest("switch "+client.getUsername()+" "+client.getCurrentBoardName()+" "+client.getCurrentBoardName());

} else {

JOptionPane.showMessageDialog(dialog, "Sorry, this username is already taken currently.", "Try again", JOptionPane.ERROR\_MESSAGE);

}

} catch (Exception e1) {

e1.printStackTrace();

}

}

});

//handles making a new board and adding it to the list

newBoardButton.addActionListener(new ActionListener() {

public synchronized void actionPerformed(ActionEvent e) {

if (!newBoard.getText().equals("")) {

NewBoardWorker worker = new NewBoardWorker(newBoard.getText());

worker.execute();

}

//if the name is empty

else {

JOptionPane.showMessageDialog(dialog, "Please enter a board name.", "Try again", JOptionPane.ERROR\_MESSAGE);

}

}

});

}

/\*\*

\* A worker to create a new board in background before adding it to the list of available boards in the beginning dialog

\*

\*/

class NewBoardWorker extends SwingWorker<Boolean, Object> {

private String newBoardName;

public NewBoardWorker(String newBoardName) {

this.newBoardName = newBoardName;

}

/\*\*

\* Called when execute is called on the worker

\*/

@Override

protected Boolean doInBackground() throws Exception {

return client.newBoard(newBoardName);

}

/\*\*

\* After doInBackground has gotten its result, display the result in the list (or not)

\*/

@Override

protected void done() {

try {

if (get()) {

String[] boards = client.getBoards();

boardListModel.removeAllElements();

for (int i=0; i<boards.length;i++) {

boardListModel.addElement(boards[i]);

}

newBoard.setText("");

} else {

JOptionPane.showMessageDialog(dialog, "Sorry, this board name is already taken.", "Try again", JOptionPane.ERROR\_MESSAGE);

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

/\*\*

\* When the user has entered their selected board, set up the canvas they will draw on an the menu

\*/

public void setupCanvas() {

this.setTitle("Whiteboard");

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setPreferredSize(new Dimension(WIDTH, HEIGHT));

this.setLayout(new BorderLayout());

this.setResizable(false);

canvas = new Canvas(client);

canvas.addDrawingController(new DrawingController(client));

this.addMenuBar();

this.addSideBar();

this.add(canvas, BorderLayout.CENTER);

this.add(sidebar, BorderLayout.EAST);

this.pack();

this.setVisible(true);

}

/\*\*

\* Creates the menubar to add to the canvas to control drawing and viewing

\*/

private void addSideBar() {

sidebar = new JPanel();

this.imageToggle = new JToggleButton(new ImageIcon(((new ImageIcon("resources/image.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.imageToggle.setPreferredSize(new Dimension(40,40));

this.imageToggle.setFocusPainted(false);

this.cropToggle = new JToggleButton(new ImageIcon(((new ImageIcon("resources/crop.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.cropToggle.setPreferredSize(new Dimension(40,40));

this.cropToggle.setFocusPainted(false);

this.square = new JToggleButton(new ImageIcon(((new ImageIcon("resources/square.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.square.setPreferredSize(new Dimension(40,40));

this.square.setFocusPainted(false);

this.triangle = new JToggleButton(new ImageIcon(((new ImageIcon("resources/triangle.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.triangle.setPreferredSize(new Dimension(40,40));

this.triangle.setFocusPainted(false);

this.circle = new JToggleButton(new ImageIcon(((new ImageIcon("resources/circle.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.circle.setPreferredSize(new Dimension(40,40));

this.circle.setFocusPainted(false);

this.brush = new JToggleButton(new ImageIcon(((new ImageIcon("resources/brushIcon.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.brush.setPreferredSize(new Dimension(40,40));

this.brush.setFocusPainted(false);

this.ereser = new JToggleButton(new ImageIcon(((new ImageIcon("resources/eraserIcon.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.ereser.setPreferredSize(new Dimension(40,40));

this.ereser.setFocusPainted(false);

this.text = new JToggleButton(new ImageIcon(((new ImageIcon("resources/text.png")).getImage())

.getScaledInstance(20, 20, java.awt.Image.SCALE\_SMOOTH)));

this.text.setPreferredSize(new Dimension(40,40));

this.text.setFocusPainted(false);

this.strokeButton = new JButton("Stroke/Text");

this.strokeButton.setFocusPainted(false);

this.strokeButton.setHorizontalAlignment(SwingConstants.LEFT);

JPanel imageBar = new JPanel();

imageBar.add(imageToggle);

imageBar.add(cropToggle);

TitledBorder imageBorder = BorderFactory.createTitledBorder("Image Editing");

imageBar.setBorder(imageBorder);

JPanel modeBar = new JPanel();

modeBar.add(brush);

modeBar.add(ereser);

TitledBorder modeBarBorder = BorderFactory.createTitledBorder("Mode");

modeBar.setBorder(modeBarBorder);

JPanel brushBar = new JPanel();

JMenuBar menuBar = new JMenuBar();

menuBar.add(getColorsMenu());

menuBar.add(getBrushTypeMenu());

brushBar.add(menuBar);

brushBar.add(strokeButton);

TitledBorder brushBarBorder = BorderFactory.createTitledBorder("Brush");

brushBar.setBorder(brushBarBorder);

JPanel textBar = new JPanel();

JMenuBar fontBar = new JMenuBar();

fontBar.add(getFontMenu());

textBar.add(text);

textBar.add(fontBar);

TitledBorder textBorder = BorderFactory.createTitledBorder("Text");

textBar.setBorder(textBorder);

JPanel shapesBar = new JPanel();

shapesBar.add(square);

shapesBar.add(triangle);

shapesBar.add(circle);

TitledBorder shapesBorder = BorderFactory.createTitledBorder("Shapes");

shapesBar.setBorder(shapesBorder);

sidebar.setLayout(new GridBagLayout());

GridBagConstraints c = new GridBagConstraints();

c.weighty = 1;

c.anchor = GridBagConstraints.NORTH;

c.insets = new Insets(5,0,0,0);

c.fill = GridBagConstraints.HORIZONTAL;

c.gridx = 0;

c.gridy = 0;

sidebar.add(modeBar, c);

c.gridx = 0;

c.gridy = 1;

sidebar.add(brushBar, c);

c.gridx = 0;

c.gridy = 2;

sidebar.add(imageBar, c);

c.gridx = 0;

c.gridy = 3;

sidebar.add(textBar, c);

c.gridx = 0;

c.gridy = 4;

sidebar.add(shapesBar, c);

strokeButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JPopupMenu popup = new JPopupMenu();

popup.add(getSlider());

popup.pack();

Point pos = new Point();

Dimension size = popup.getPreferredSize();

pos.x = (strokeButton.getWidth()/2 - size.width/2);

pos.y = (strokeButton.getHeight());

popup.show(strokeButton, pos.x, pos.y);

}

});

imageToggle.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

setImage(e);

}

});

square.addItemListener(new ItemListener() {

@Override

public void itemStateChanged(ItemEvent e) {

if (square.isSelected()) {

brush.setSelected(false);

ereser.setSelected(false);

text.setSelected(false);

client.setSquareOn(true);

client.setEraserOn(false);

client.setBrushOn(false);

client.setText(false);

System.out.println("square press enable" + Boolean.toString(client.getSquareOn()) );

}

else {

square.setEnabled(true);

client.setSquareOn(false);

System.out.println("square press disable" + Boolean.toString(client.getSquareOn()) );

}

}

});

ereser.addItemListener(new ItemListener() {

@Override

public void itemStateChanged(ItemEvent e) {

if (ereser.isSelected()) {

square.setSelected(false);

brush.setSelected(false);

text.setSelected(false);

client.setEraserOn(true);

client.setBrushOn(true);

client.setSquareOn(false);

client.setText(false);

}

else {

client.setBrushOn(false);

}

}

});

brush.addItemListener(new ItemListener() {

@Override

public void itemStateChanged(ItemEvent e) {

if (brush.isSelected()) {

client.setStraight(true);

square.setSelected(false);

text.setSelected(false);

ereser.setSelected(false);

client.setEraserOn(false);

client.setBrushOn(true);

client.setSquareOn(false);

client.setText(false);

}

else {

client.setBrushOn(false);

}

}

});

text.addItemListener(new ItemListener() {

@Override

public void itemStateChanged(ItemEvent e) {

if (text.isSelected()) {

client.setStraight(true);

square.setSelected(false);

ereser.setSelected(false);

client.setEraserOn(false);

client.setBrushOn(false);

client.setText(true);

client.setSquareOn(false);

}

else {

client.setText(false);

}

}

});

}

private void setImage(ActionEvent e) {

{

JFileChooser file = new JFileChooser();

file.setCurrentDirectory(new File(System.getProperty("user.home")));

//filter the files

FileNameExtensionFilter filter = new FileNameExtensionFilter("\*.Images", "jpg","gif","png");

file.addChoosableFileFilter(filter);

int result = file.showSaveDialog(null);

//if the user click on save in Jfilechooser

if(result == JFileChooser.APPROVE\_OPTION){

File selectedFile = file.getSelectedFile();

String path = selectedFile.getAbsolutePath();

canvas.fillWithImageAndCall(path);

}

//if the user click on save in Jfilechooser

else if(result == JFileChooser.CANCEL\_OPTION){

System.out.println("No File Select");

}

}

}

private void addMenuBar() {

JMenuBar menuBar = new JMenuBar();

menuBar.add(getUsersMenu());

menuBar.add(getBoardsMenu());

menuBar.add(getSaveMenu());

menuBar.add(Box.createHorizontalGlue());

currentUserBoard = getCurrentUserBoard();

menuBar.add(currentUserBoard);

menuBar.add(Box.createHorizontalGlue());

this.setJMenuBar(menuBar);

}

private JMenu getSaveMenu() {

final JMenu mode = new JMenu("Save");

JMenuItem drawMenuItem = new JMenuItem("Save As...");

drawMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

canvas.saveMethod();

}});

mode.add(drawMenuItem);

return mode;

}

/\*\*

\* Add the users menu to the menu mar

\* @return JMenu representing the users menu

\*/

private JMenu getUsersMenu() {

final JMenu usersMenu = new JMenu("Users");

//List of Users

try {

for (String user: client.getUsers()) {

JLabel label = new JLabel(user);

label.setBorder(BorderFactory.createEmptyBorder(2, 5, 3, 5));

usersMenu.add(label);

}

} catch (Exception e) {

e.printStackTrace();

}

usersMenu.addMenuListener(new MenuListener() {

@Override

public void menuCanceled(MenuEvent arg0) {

}

@Override

public void menuDeselected(MenuEvent arg0) {

}

@Override

public void menuSelected(MenuEvent arg0) {

//refresh the users list whenever the menu button is pressed

usersMenu.removeAll();

try {

for (String user: client.getUsers()) {

JLabel label = new JLabel(user);

label.setBorder(BorderFactory.createEmptyBorder(2, 5, 3, 5));

usersMenu.add(label);

}

} catch (Exception e) {

e.printStackTrace();

}

}

});

return usersMenu;

}

/\*\*

\* Add the boards menu to the menu mar

\* @return JMenu representing the boards menu

\*/

private JMenu getBoardsMenu() {

//add List of Boards

final JMenu boards = new JMenu("Board(s)");

//clicking the new board button brings up the new board dialog

JMenuItem newBoardButton = new JMenuItem("New Board");

boards.add(newBoardButton);

newBoardButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

client.getClientGUI().newBoardDialog();

}

});

boards.addSeparator();

//List of Boards

String[] listBoards = {};

try {

listBoards = client.getBoards();

} catch (Exception e) {

e.printStackTrace();

}

//when a board is clicked, the client should switch the current board to that board

for (final String board: listBoards) {

JMenuItem boardChoice = new JMenuItem(board);

boardChoice.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

canvas.switchBoard(board);

}

});

boards.add(boardChoice);

}

boards.addMenuListener(new MenuListener() {

@Override

public void menuCanceled(MenuEvent arg0) {

}

@Override

public void menuDeselected(MenuEvent arg0) {

}

@Override

public void menuSelected(MenuEvent arg0) {

//refresh the boards list whenever the boards list is clicked

for (int i=boards.getItemCount()-1; i>1; i--) {

boards.remove(i);

}

try {

for (final String board: client.getBoards()) {

JMenuItem boardChoice = new JMenuItem(board);

boardChoice.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

canvas.switchBoard(board);

}

});

boards.add(boardChoice);

}

} catch (Exception e) {

e.printStackTrace();

}

}

});

return boards;

}

/\*\*

\* Add the mode menu to the menu mar

\* @return JMenu representing the mode menu

\*/

private JMenu getFontMenu() {

final JMenu type = new JMenu("Font");

JMenuItem arialMenuItem = new JMenuItem("Arial");

arialMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

client.setFontArial(true);

client.setFontComic(false);

}});

JMenuItem comicMenuItem = new JMenuItem("Comic Sans");

comicMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

client.setFontArial(false);

client.setFontComic(true);

}});

type.add(arialMenuItem);

type.addSeparator();

type.add(comicMenuItem);

type.setBorder(BorderFactory.createLineBorder(Color.BLACK,2));

return type;

}

/\*\*

\* Add the brushtype menu to the menu mar

\* @return JMenu representing the brushtype menu

\*/

private JMenu getBrushTypeMenu() {

final JMenu type = new JMenu("Type");

JMenuItem straightMenuItem = new JMenuItem("Straight");

straightMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

client.setStraight(true);

System.out.println("straight on");

client.setDotted(false);

client.setDashed(false);

}});

JMenuItem dottedMenuItem = new JMenuItem("Dotted");

dottedMenuItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent event) {

client.setStraight(false);

client.setDotted(true);

client.setDashed(false);

}});

type.add(straightMenuItem);

type.addSeparator();

type.add(dottedMenuItem);

type.setBorder(BorderFactory.createLineBorder(Color.BLACK,2));

return type;

}

/\*\*

\* Add the colors menu to the menu bar

\* @return JMenu representing the colors menu

\*/

private JMenu getColorsMenu() {

class ColorChangeListener implements ChangeListener {

JMenu colors;

public ColorChangeListener(JMenu colors) {

this.colors = colors;

}

@Override

public void stateChanged(ChangeEvent e) {

ColorSelectionModel model = (ColorSelectionModel) e.getSource();

Color currentColor = model.getSelectedColor();

client.setCurrentColor(currentColor);

colors.setBorder(BorderFactory.createLineBorder(currentColor,2));

}

}

//add Colors

JMenu colors = new JMenu("Paint Color");

JColorChooser chooser = new JColorChooser(client.getCurrentColor());

colors.add(chooser);

chooser.getSelectionModel().addChangeListener(new ColorChangeListener(colors));

chooser.setPreviewPanel(new JPanel());

colors.setBorder(BorderFactory.createLineBorder(client.getCurrentColor(),2));

//remove panels

AbstractColorChooserPanel[] panels = chooser.getChooserPanels();

for (AbstractColorChooserPanel accp : panels) {

if (!accp.getDisplayName().equals("Swatches")) {

chooser.removeChooserPanel(accp);

}

}

return colors;

}

/\*\*

\* add slider to the menu bar

\* @return JSlider representing the slider

\*/

private JSlider getSlider() {

class SliderChangeListener implements ChangeListener {

public void stateChanged(ChangeEvent e) {

JSlider source = (JSlider)e.getSource();

if (!source.getValueIsAdjusting()) {

float weight = (float)source.getValue();

client.setCurrentWidth(weight);

}

}

}

JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, (int)Math.round(client.getCurrentWidth()));

slider.addChangeListener(new SliderChangeListener());

slider.setMajorTickSpacing(10);

slider.setMinorTickSpacing(2);

slider.setPaintTicks(true);

slider.setPaintLabels(true);

slider.setVisible(true);

return slider;

}

/\*\*

\* The label to tell the user what their username is and which board they're using

\* @return

\*/

public JLabel getCurrentUserBoard() {

String user = client.getUsername();

String board = client.getCurrentBoardName();

return new JLabel("Hi, " + user + ". This board is: " + board);

}

/\*\*

\* The dialog for creating a new board when in canvas view

\*/

public void newBoardDialog() {

final JDialog newBoardDialog = new JDialog();

newBoardDialog.setTitle("Create New Board");

newBoardDialog.setResizable(false);

final Container newBoardDialogContainer = new Container();

GroupLayout layout = new GroupLayout(newBoardDialogContainer);

layout.setAutoCreateGaps(true);

layout.setAutoCreateContainerGaps(true);

newBoardDialogContainer.setLayout(layout);

JLabel newBoardNameLabel = new JLabel("New Board Name:");

final JTextField newBoardName = new JTextField(10);

JButton newBoardButton = new JButton("Create");

ParallelGroup hGroup = layout.createParallelGroup(GroupLayout.Alignment.CENTER);

SequentialGroup hEnter = layout.createSequentialGroup();

hEnter.addComponent(newBoardNameLabel).addComponent(newBoardName);

hGroup.addGroup(hEnter).addComponent(newBoardButton);

ParallelGroup vGroup = layout.createParallelGroup(GroupLayout.Alignment.LEADING);

SequentialGroup vAll = layout.createSequentialGroup();

ParallelGroup v1 = layout.createParallelGroup(GroupLayout.Alignment.BASELINE);

v1.addComponent(newBoardNameLabel).addComponent(newBoardName);

vAll.addGroup(v1).addComponent(newBoardButton);

vGroup.addGroup(vAll);

layout.setHorizontalGroup(hGroup);

layout.setVerticalGroup(vGroup);

newBoardDialog.setContentPane(newBoardDialogContainer);

newBoardDialog.pack();

newBoardDialog.setVisible(true);

newBoardButton.addActionListener(new ActionListener() {

public synchronized void actionPerformed(ActionEvent e) {

String newBoardNameString = newBoardName.getText();

if (newBoardNameString.equals("")) {

JOptionPane.showMessageDialog(newBoardDialog, "Please enter a board name.", "Try again", JOptionPane.ERROR\_MESSAGE);

} else {

try {

boolean successful = client.newBoard(newBoardNameString);

if (!successful) {

JOptionPane.showMessageDialog(newBoardDialog, "Sorry, this board name is already taken.", "Try again", JOptionPane.ERROR\_MESSAGE);

} else {

client.getBoards();

newBoardDialog.dispose();

}

} catch (Exception e1) {

e1.printStackTrace();

}

}

}

});

}

/\*\*

\* Resets the label that tells the user what their username and current board are

\* @param newBoard: the new label to set the label to

\*/

public void setCurrentUserBoard(JLabel newBoard) {

currentUserBoard = newBoard;

}

/\*\*

\* Gets the canvas, a controller, for the GUI

\* @return the canvas object

\*/

public Canvas getCanvas() {

return canvas;

}

}

\*\*\*\*\*\*\*\*\*ClientReceiveProtocol.java\*\*\*\*\*\*\*\*\*\*\*\*

package client;

import java.io.BufferedReader;

import java.io.IOException;

import command.Command;

public class ClientReceiveProtocol implements Runnable {

private final BufferedReader in;

private final Client client;

private boolean isRunning = true;

public ClientReceiveProtocol(BufferedReader in, Client client) {

this.in= in;

this.client = client;

}

/\*\*

\* Waits for message from server and calls appropriate request handler

\*/

@Override

public void run() {

// provide a way to kill thread

while(isRunning) {

//handle the client

try {

handleConnection(in);

} catch (IOException e) {

// Means connection has closed

}

}

}

/\*\*

\* Handle connection to server. Returns when client disconnects.

\*

\* @param socket socket where the client is connected

\* @throws IOException if connection has an error or terminates unexpectedly

\*/

private void handleConnection(BufferedReader in) throws IOException {

for (String line = in.readLine(); line != null; line = in.readLine()) {

handleRequest(line);

}

}

/\*\*

\* Handler for server input, performing requested operations and returning an output message.

\* Receives:

\*

\* Update Users = "users boardName user1 user2 user3..."

\* Update Available Boards = "boards board1 board2 board3"

\* Draw = "draw boardName command param1 param2 param3"

\* Example: "draw boardName drawLineSegment x1 y1 x2 y2 color width"

\* Check and add User = "checkAndAddUser username boardName boolean"

\* New Board = "newBoard boardName boolean"

\*

\* @param input message from server

\* @return message to client

\* @throws IOException

\*/

private void handleRequest(String input) throws IOException, IllegalArgumentException {

String nameReg = "[a-zA-Z0-9\\.\\:\\\\\\[\\]\\=\\,]+";

String regex = "(draw "+nameReg+"( "+nameReg+")+)|"

+ "(users( "+nameReg+")+)|"

+ "(exit "+nameReg+")|"

+"(boards( "+nameReg+")\*)|"

+ "(checkAndAddUser ("+nameReg+" "+nameReg+" (true|false)))|"

+"(newBoard "+nameReg+" (true|false))|"

+ "(switch "+nameReg+" "+nameReg+" "+nameReg+")";

// make sure it's a valid input

if (input.matches(regex)) {

try {

String[] tokens = input.split(" ");

//take the boards from the response and set them to the list of boards

if (tokens[0].equals("boards")) {

client.setBoards(client.parseBoardsFromServerResponse(input));

}

//parse from the response whether the board has been created

else if (tokens[0].equals("newBoard")) {

client.parseNewBoardFromServerResponse(input);

}

//parse from the response whether the new user has entered successfully

else if (tokens[0].equals("checkAndAddUser")) {

client.parseNewUserFromServerResponse(input);

}

//is the set of users is for the correct board

//parse the users from the response and set them to the list of users

else if (tokens[0].equals("users")) {

if (client.checkForCorrectBoard(tokens[1])) {

client.setUsers(client.parseUsersFromServerResponse(input));

}

}

//when the response is received,the client has exited the server and the threads can be stopped

else if (tokens[0].equals("exit")) {

client.completeExit();

}

//check that the draw command is for this board

//invoke the command received on the client's canvas

else if (tokens[0].equals("draw")) {

Command command = new Command(input.split(" "));

if (command.checkBoardName(client.getCurrentBoardName())) {

client.applyCommand(command);

}

}

} catch (Exception e) {

e.printStackTrace();

}

} else {

System.out.println("Invalid response");

}

}

/\*\*

\* Used to kill thread from outside

\*/

public void kill() {

isRunning = false;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ClientSendProtocol.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package client;

import java.io.IOException;

import java.io.PrintWriter;

/\*\*

\* Asynchronous protocol to send messages out over a PrintWriter socket

\* @author Josh

\*

\*/

public class ClientSendProtocol implements Runnable {

private final PrintWriter out;

private final String message;

/\*\*

\* Asynchronous printwriter. Writes message to PrintWriter socket.

\* @param out: PrintWriter to write message to

\* @param message: message to write

\*/

public ClientSendProtocol(PrintWriter out, String message) {

this.out = out;

this.message = message;

}

/\*\*

\* Sends message to server over a PrintWriter

\* @throws IOException

\*/

@Override

public void run() {

out.println(message);

}

}

\*\*\*\*\*\*\*\*\*DrawingController.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package client;

import java.awt.Color;

import java.awt.Point;

import java.awt.Shape;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.awt.event.MouseMotionListener;

import java.util.ArrayList;

/\*

\* DrawingController handles the user's freehand drawing.

\*/

public class DrawingController implements MouseListener, MouseMotionListener {

// store the coordinates of the last mouse event, so we can

// draw a line segment from that last point to the point of the next mouse event.

private int lastX, lastY;

private final Client client;

// coordinates for drawing a square

private int x, y, x2, y2;

// time for dotted line

private int i = 0;

private boolean drawOn = true;

ArrayList<Shape> shapes = new ArrayList<Shape>();

Point startDrag, endDrag;

public DrawingController(Client client) {

this.client = client;

}

public void setStartPoint(int x, int y) {

this.x = x;

this.y = y;

}

public void setEndPoint(int x, int y) {

x2 = (x);

y2 = (y);

}

/\*

\* When mouse button is pressed down, start drawing.

\*/

public void mousePressed(MouseEvent e) {

System.out.println("square " + Boolean.toString(client.getSquareOn()) );

System.out.println("text" + Boolean.toString(client.getText()));

if(client.getSquareOn() == false) {

if(client.getText()==true)

{

lastX = e.getX();

lastY = e.getY();

String textFont = new String();

if(client.getFontArial() == true)

textFont = "Arial";

else

if(client.getFontComic() == true)

textFont = "Comic Sans MS";

Color color = client.getCurrentColor();

client.getCanvas().drawText(textFont, color.getRGB(),(int)client.getCurrentWidth(), 0, lastX, lastY);

}

else

if(client.getBrushOn()==true) {

lastX = e.getX();

lastY = e.getY();

}

}

else

{

setStartPoint(e.getX(), e.getY());

startDrag = new Point(e.getX(), e.getY());

endDrag = startDrag;

client.getCanvas().newDrawSquare(startDrag, endDrag);

}

}

/\*

\* When mouse moves while a button is pressed down,

\* draw a line segment or a square

\*/

public void mouseDragged(MouseEvent e) {

if(client.getSquareOn() == false) {

if(client.getBrushOn()==true) {

int x = e.getX();

int y = e.getY();

Color color = client.getCurrentColor();

if (client.getEraserOn()) { color = Color.white; client.setStraight(true);}

if(client.getStraight()==true) {

client.getCanvas().drawLineSegmentAndCall(lastX, lastY, x, y, color.getRGB(), client.getCurrentWidth());

System.out.println("is straight");

}

else

if(client.getDotted()) {

i++;

if(i == 30)

{

if(drawOn == false)

drawOn = true;

else

drawOn = false;

i = 0;

}

if(drawOn == true)

client.getCanvas().drawLineSegmentAndCall(lastX, lastY, x, y, color.getRGB(), client.getCurrentWidth());

System.out.println("is dotted");

}

lastX = x;

lastY = y;

}

else {

endDrag = new Point(e.getX(), e.getY());

setEndPoint(e.getX(), e.getY());

//client.getCanvas().newDrawSquare(startDrag, endDrag);

}

}

}

/\*

\* When mouse is realeased draw a rectangle

\*/

public void mouseReleased(MouseEvent e) {

if(client.getSquareOn() == true) {

Color color = client.getCurrentColor();

setEndPoint(e.getX(), e.getY());

client.getCanvas().drawSquareAndCall(x, y, x2, y2, color.getRGB(), client.getCurrentWidth());

startDrag = null;

endDrag = null;

}

}

public void mouseMoved(MouseEvent e) { }

public void mouseClicked(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

}

\*\*\*\*\*\*\*\*\*\*\*\*Tracker.java\*\*\*\*\*\*\*\*\*\*\*\*\*

package client;

/\*\*

\* Mutable boolean object to allow tracking flags to be passed by reference

\* @author Josh

\*

\*/

public class Tracker {

private boolean value;

/\*\*

\* Initializes flag to value

\* @param value

\*/

public Tracker(boolean value) {

this.value = value;

}

/\*\*

\* Setter

\* @param value

\*/

public synchronized void setValue(boolean value){

this.value = value;

}

/\*\*

\* Getter

\* @return

\*/

public synchronized boolean getValue() {

return value;

}

}

\*\*\*\*\*\*\*\*\*\*\*Command.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

package command;

import java.awt.Color;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

import java.util.Arrays;

import client.Canvas;

/\*\*

\* Creates command from token array passed that has already been determined to be a draw command

\* @param elements: Elements of command in format ["draw", "boardName", "command", "arg1", "arg2", "arg3", ...]

\* @return a Command object with the command and the arguments

\*/

public class Command {

private final String command;

private final String[] arguments;

private final String boardName;

public Command(String[] elements) {

String[] arguments = new String[elements.length-3];

for (int i=3; i<elements.length;i++) {

arguments[i-3] = elements[i];

}

this.command = elements[2];

this.boardName = elements[1];

this.arguments = arguments;

}

/\*\*

\* Straightforwardly takes the given parameters and makes a command object out of them

\* @param boardName: name of the board the command is for

\* @param command: name of the command

\* @param arguments: the list of arguments as strings

\*/

public Command(String boardName, String command, String[] arguments) {

this.boardName = boardName;

this.command = command;

this.arguments = arguments;

}

/\*\*

\* Finds the method with a name matching the command name,

\* then invokes the method with the command's arguments

\* @param canvas: the object that the method will be invoked on

\*/

public void invokeCommand(Canvas canvas) {

Method[] methods = Canvas.class.getMethods();

Method method = null;

for (int i=0; i<methods.length;i++) {

if (methods[i].getName().equals(command)) {

method = methods[i];

System.out.println(methods[i].getName());

}

}

if (method == null) {

throw new RuntimeException("Command "+command+" not found.");

} else {

Class<?>[] parameters = method.getParameterTypes();

if (parameters.length != arguments.length) {

System.out.println("parameters" + parameters.length);

for (int i=0; i<parameters.length;i++) {

System.out.println(parameters[i].getName());

}

System.out.println("argumets" + arguments.length);

for (int i=0; i<arguments.length;i++) {

System.out.println(arguments[i]);

}

throw new RuntimeException("Incorrect number of arguments for given method.");

} else {

Object[] typedArgs = new Object[arguments.length];

for (int i=0; i<typedArgs.length;i++) {

if (parameters[i].equals(int.class)) {

typedArgs[i] = Integer.valueOf(arguments[i]);

} else if(parameters[i].equals(float.class)) {

typedArgs[i] = Float.valueOf(arguments[i]);

} else if(parameters[i].equals(Color.class)) {

typedArgs[i] = Color.decode(arguments[i]);

} else if(parameters[i].equals(double.class)) {

typedArgs[i] = Double.valueOf(arguments[i]);

} else if(parameters[i].equals(long.class)) {

typedArgs[i] = Long.valueOf(arguments[i]);

} else if(parameters[i].equals(boolean.class)) {

typedArgs[i] = Boolean.valueOf(arguments[i]);

} else if(parameters[i].equals(short.class)) {

typedArgs[i] = Float.valueOf(arguments[i]);

} else if(parameters[i].equals(byte.class)) {

typedArgs[i] = Byte.valueOf(arguments[i]);

} else if(parameters[i].equals(char.class)) {

typedArgs[i] = arguments[i].charAt(0);

} else {

typedArgs[i] = parameters[i].cast(arguments[i]);

}

}

try {

method.invoke(canvas, typedArgs);

} catch (IllegalAccessException | IllegalArgumentException

| InvocationTargetException e) {

e.printStackTrace();

}

}

}

}

/\*\*

\* Compares the board name given to make sure the command is for the same board

\* @param compareBoardName: the board name the command should be for

\* @return whether or not this command is for the board name given

\*/

public boolean checkBoardName(String compareBoardName) {

return this.boardName.equals(compareBoardName);

}

@Override

public String toString() {

StringBuilder argumentString = new StringBuilder(" ");

for (String arg : arguments) {

argumentString.append(arg+" ");

}

argumentString.deleteCharAt(argumentString.length()-1);

return "draw "+boardName+" "+command+argumentString;

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof Command)) return false;

Command commandObj = (Command) obj;

return commandObj.command.equals(command) && Arrays.equals(commandObj.arguments, arguments) && commandObj.boardName.equals(boardName);

}

}