Portfolio 1
CS 319

Communication Project with Painting Functionality

Team Member: Zijie Deng, Cheng Song

Table of Contents

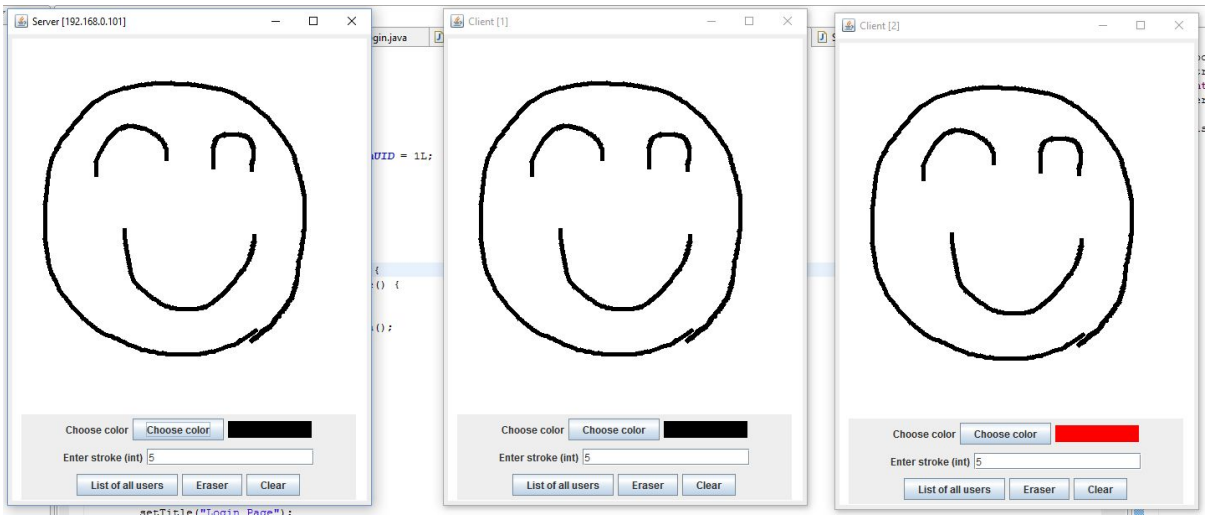
Introduction:

Chatting App is becoming an indispensable part in our social life. People start using Chatting App instead of sending text or using other kinds of communication software. Some of the powerful Chatting Apps can send messages, emoji, pictures, voice records or even videos, but all these communication methods require user waiting for another user to tap the send button. Our team believe that if two users can write or draw picture on a single page like two people using a single painting board, the interaction between two users will be much higher than texting messages. This communication method may be used by close friends or couples to improve their relationships. Based on this point, we decide to design a software which allows users can draw synchronously.

To reach this goal, we have done this following steps:

1. We used Java Swing to create the user interface. Swing is a GUI widget toolkit for Java. It provides a native look and feel that emulates the look and feel of several platforms, and supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.
2. We used Java establish server and clients
3. We used MouseListener and MouseMotionListener to track what clients drawing
4. We used ObjectStream to transfer drawn lines and paint context (e.g. color, stroke, who draws it)

Follow figure shows an example how does our project look like:



New & Complex

1. **Draw Synchronously:**

In our project, client will register as paint listener to receive paint event, whenever there is a paint event, a new thread is started to send that event to server, and server will broadcast the paint event to other clients synchronously.

Synchronously is the most important part of this project. In homework 1, we did create a project can receive and send characters and pictures. But all the messages have been sent and received are after they are completely finished by the sending client. For our project, we can transfer every line created by client synchronously to server and other clients. Users can see the entire drawing process and this function can let multiple users cooperate with each other to accomplish a single painting. Aiming to this goal, we used threads and ObjectStream to transfer the drawn lines.

Object Streams:

Object Streams support I/O of objects.

The object stream classes are ObjectInputStream and ObjectOutputStream. These classes implement ObjectInput and ObjectOutput, which are subinterfaces of DataInput and DataOutput. That means that all the primitive data I/O methods covered in Data Streams are also implemented in object streams. So an object stream can contain a mixture of primitive and object values.

2. **Draw in different color and different thickness**

In our project, clients can draw in different color and different thickness. These following figures show how the painting looks like.

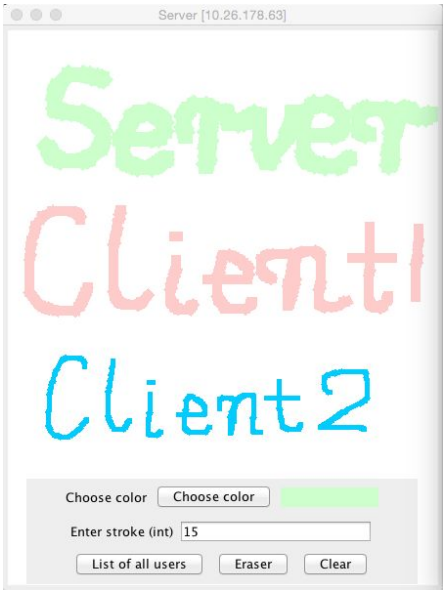


Figure.1

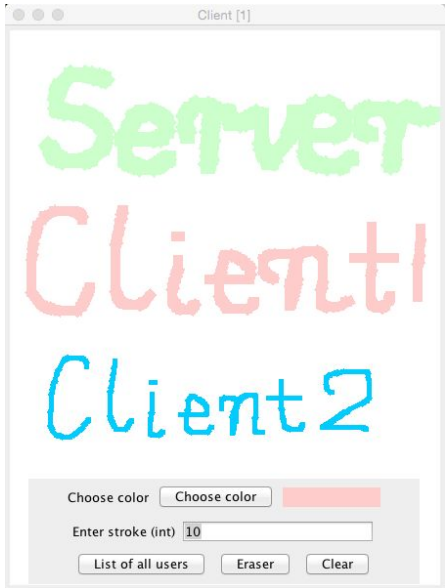


Figure. 2

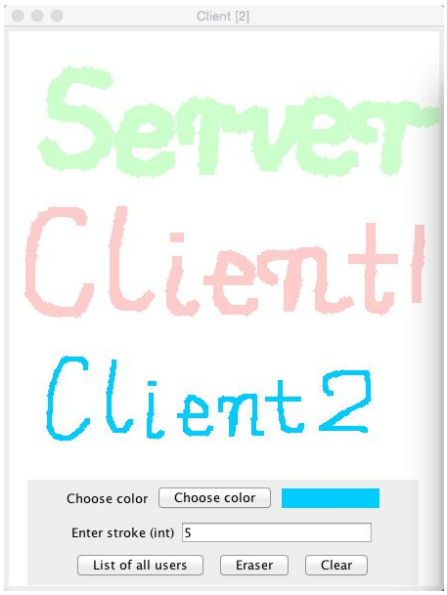
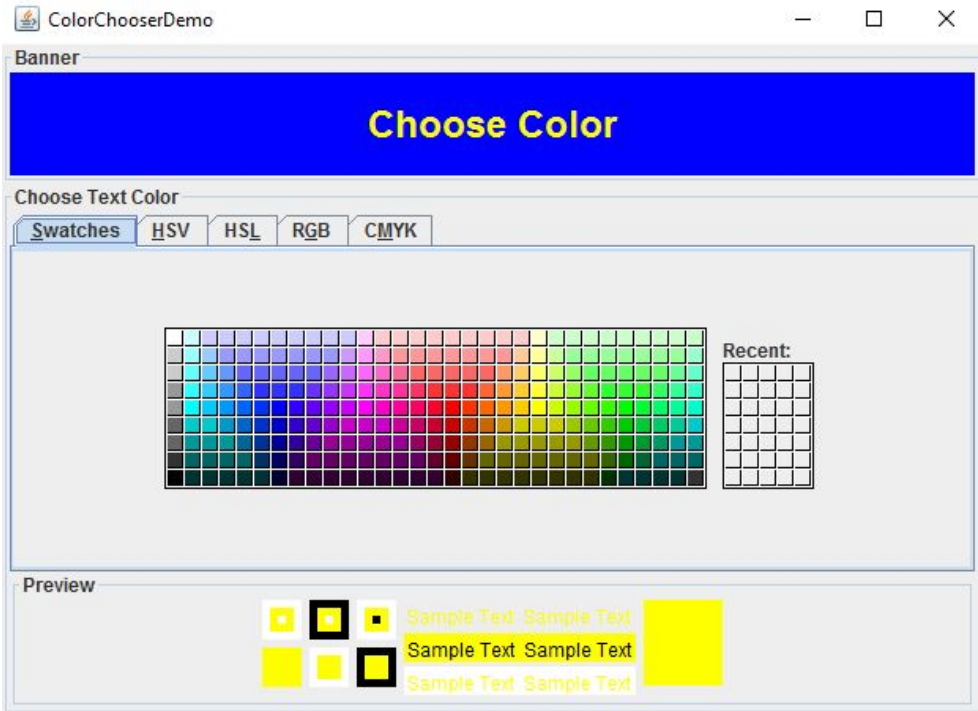


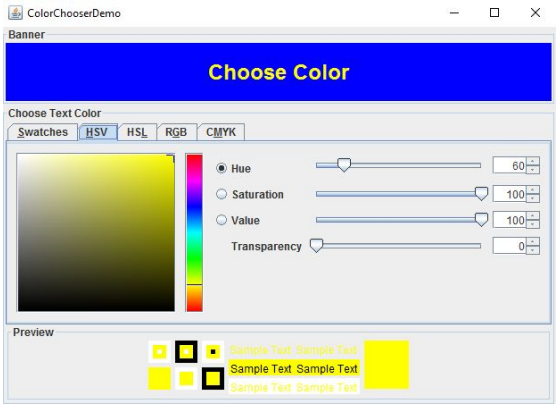
Figure. 3

Figure.1 shows how we wrote “Server” in green and the stroke thickness is 15
Figure.2 shows how we wrote “Client1” in pink and the stroke thickness is 10
Figure.3 shows how we wrote “Client2” in blue and the stroke thickness is 5

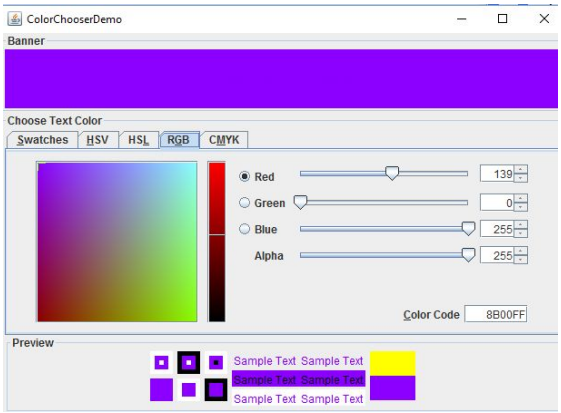
To change the thickness, we have to set the stroke. An example shows in this following figure:

To draw different colors, we use a color panel adapted from [source: https://docs.oracle.com/javase/tutorial/uiswing/components/colorchooser.html]. Panel looks the same in these following figures:

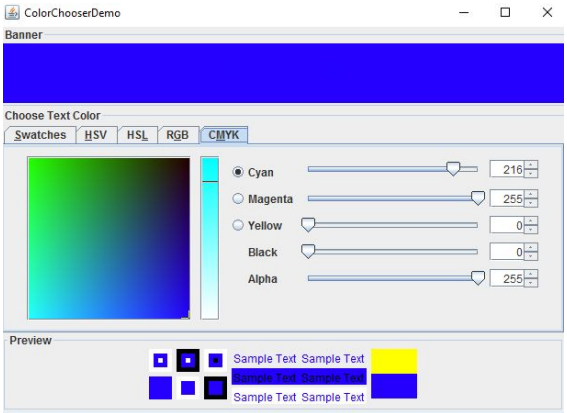




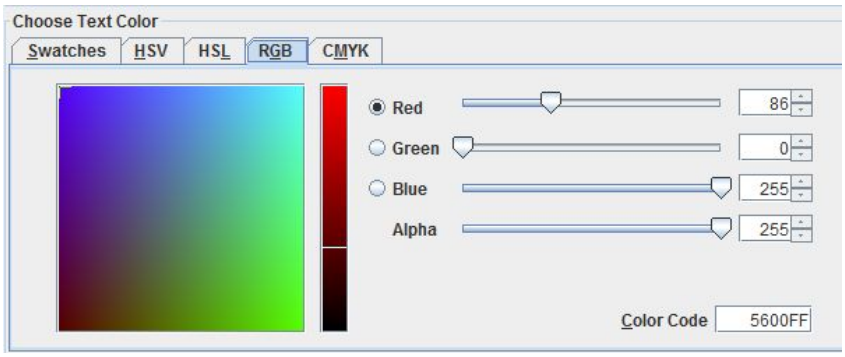
HSV can change value of Hue, Saturation, Value and Transparency



HSL can change additional lightness value

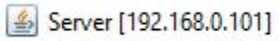


CMYK can change values of Cyan, Magenta, Yellow, Black and Alpha



RGB can mix red, green and blue.
With all these kinds of panels, we can get almost every visible color in the world.

3. IP address shows on server:



Like the above figure shows, Server can show the Server's IP address. And when every time client login, client can type its IP address into program.

Bloom's Taxonomy

1. Receive painting

Here is the method of clients receiving the drawn lines and set up events. One thread is responsible for receiving such events. And we used Object Stream to transfer the drawn lines.

```
public void run() {
    printSocketInfo(s);
    try {
        ObjectInputStream ois = new ObjectInputStream(s.getInputStream());
        while (true) {
            while (true) {
                Object o = null;
                try {
                    o = ois.readObject();
                } catch (Exception e) {
                }
                if (o instanceof PaintContext) {
                    PaintContext pc = (PaintContext) o;
                    if (pc.sw == PaintContext.Switch.DRAW_LINE) {
                        pc.drawLineOn(frame);
                        Server.server.drawLine(pc.x, pc.y, pc.nx, pc.ny);
                    } else {
                        pc.setUpOn(frame);
                        Server.allusers.addAll(pc.username);
                        Server.server.setUp(pc.color, pc.stroke);
                    }
                }
            }
        }

    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

2. Send painting

The following code is the method of server sending the paint context to all clients, which also using Object Streams.

```
private void addThread(Socket socket) {
    // add new client
    Thread t = new Thread(new ClientHandler(socket, clientNum, frame));
    t.start();
    sockets.add(socket);
    try {
        ooss.add(new ObjectOutputStream(socket.getOutputStream()));
    } catch (IOException e) {
    }
}

public void drawLine(int x, int y, int nx, int ny) {
    for (ObjectOutputStream oos : ooss) {
        try {
            oos.writeObject(new PaintContext(x, y, nx, ny));
            oos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void setUp(Color color, int stroke) {
    for (ObjectOutputStream oos : ooss) {
        try {
            oos.writeObject(new PaintContext(color, stroke, allusers));
            oos.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

3. Drawing

Color Choosing Panel:

We showed the fantastic color choosing panel in the above figures. For doing that, we used JColorChooeser class to enable from a palette of colors. The color chooser consists of everything within the box labeled Choose Text Color. This is what a standard color chooser looks like in the Java Look & Feel. It contains two parts, a tabbed pane and a preview panel. The three tabs in the tabbed pane select *chooser panels*. The *preview panel* below the tabbed pane displays the currently selected color.

```
class ColorChooserDemo extends JPanel implements ChangeListener {

    private static final long serialVersionUID = 1L;
    protected JColorChooser tcc;
    protected JLabel banner, bannerExt;

    public ColorChooserDemo(JLabel bannerExt) {
        super(new BorderLayout());

        // Set up the banner at the top of the window

        banner = new JLabel("Choose Color", JLabel.CENTER);
        banner.setForeground(Color.yellow);
        banner.setBackground(Color.blue);
        banner.setOpaque(true);
        banner.setFont(new Font("SansSerif", Font.BOLD, 24));
        banner.setPreferredSize(new Dimension(100, 65));

        JPanel bannerPanel = new JPanel(new BorderLayout());
        bannerPanel.add(banner, BorderLayout.CENTER);
        bannerPanel.setBorder(BorderFactory.createTitledBorder("Banner"));

        // Set up color chooser for setting text color
        tcc = new JColorChooser(banner.getForeground());
        tcc.getSelectionModel().addChangeListener(this);
        tcc.setBorder(BorderFactory.createTitledBorder("Choose Text Color"));

        add(bannerPanel, BorderLayout.CENTER);
        add(tcc, BorderLayout.PAGE_END);

        this.bannerExt = bannerExt;
    }

    public void stateChanged(ChangeEvent e) {
        Color newColor = tcc.getColor();
        banner.setForeground(newColor);
        banner.setBackground(newColor);

        bannerExt.setForeground(newColor);
        bannerExt.setBackground(newColor);
    }
```

Drawing Lines:

For drawing the lines, we used mouse listener to track the stroke. The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the MouseMotionListener.)

Based on different clients can draw together, we have to make sure thread safety. For thread safety, this following method should be invoked from the event-dispatching thread by using javax.swing.SwingUtilities.invokeLater

```
/**
 * Create the GUI and show it. For thread safety, this method should be
 * invoked from the event-dispatching thread.
 */
private static void createAndShowGUI(JLabel banner) {
    // Create and set up the window.
    JFrame frame = new JFrame("ColorChooserDemo");
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

    // Create and set up the content pane.
    JComponent newContentPane = new ColorChooserDemo(banner);
    newContentPane.setOpaque(true); // content panes must be opaque
    frame.setContentPane(newContentPane);

    // Display the window.
    frame.pack();
    frame.setVisible(true);
}

public static void chooseColor(JLabel banner) {
    // Schedule a job for the event-dispatching thread:
    // creating and showing this application's GUI.
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            createAndShowGUI(banner);
        }
    });
}
```

This following code shows how the lines be drawn by clients or server.

```java
import java.awt.Color;

public class PaintContext implements Serializable {

    private static final long serialVersionUID = 1L;

    int x, y, nx, ny;

    Switch sw;

    Color color;

    int stroke;

    HashSet<String> username = new HashSet<>();

    public PaintContext(int x, int y, int nx, int ny) {
        this.x = x;
        this.y = y;
        this.nx = nx;
        this.ny = ny;

        sw = Switch.DRAW_LINE;
    }

    public PaintContext(Color color, int stroke, String username) {
        this.color = color;
        this.stroke = stroke;
        this.username.add(username);

        sw = Switch.SET_UP;
    }

    public PaintContext(Color color, int stroke, HashSet<String> username) {
        this.color = color;
        this.stroke = stroke;
        this.username.addAll(username);

        sw = Switch.SET_UP;
    }

    public void drawLineOn(Paintable p) {
        p.drawLine(x, y, nx, ny);
    }

    public void setUpOn(Paintable p) {
        p.setUp(color, stroke);
    }

    enum Switch {
        DRAW_LINE, SET_UP
    }
```

In the Client Class, we registered PaintListener to track the paint:

```
public void start() throws IOException {
    frame = new ClientGUI(username, new PaintListener() {
        @Override
        public void drawLine(int x, int y, int nx, int ny) {
            new Thread() {
```

In the Server Class, we also registered PaintListener to track the paint:

```
private PaintListener paintListener = new PaintListener() {

    @Override
    public void drawLine(int x, int y, int nx, int ny) {
        Server.this.drawLine(x, y, nx, ny);
    }
```

Following code is PaintListener, which is used to detect the paint:

```
import java.awt.Color;

public interface PaintListener {
    void drawLine(int x, int y, int nx, int ny);

    void setUp(Color color, int stroke);
}
```

And to make the program being drawable, we implemented Paintable in ServerGUI and ClientGUI:

```
public class ServerGUI extends JFrame implements Paintable {
```

```
public class ClientGUI extends JFrame implements Paintable {
```

Following code is Paintable, which makes the GUI to be drawn:

```
import java.awt.Color;

public interface Paintable {

    void drawLine(int x, int y, int nx, int ny);

    void setUp(Color c, int stroke);
}
```

IP address of server and clients:

This following code has a function can capture the IP Address:

```
public void start() {
    try {
        frame = new ServerGUI(paintListener, InetAddress.getLocalHost().getHostAddress());
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
    frame.setVisible(true);
    ((PaintPanel) frame.pPanel).users = new PaintUsers() {
        @Override
        public HashSet<String> allusers() {
            return allusers;
        }
    };
}
```

And this following code is building for typing IP address of clients into the program

```
public Client(String ipAddr, String username, int serverPort) {
    this.username = username;

    // set up the socket to connect to the gui
    try {
        socket = new Socket(ipAddr, serverPort);
        start();
    } catch (UnknownHostException h) {
        JOptionPane.showMessageDialog(new JFrame(), "Unknown Host " + h.getMessage());
        System.exit(1);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(new JFrame(), "IO exception: " + e.getMessage());
        System.exit(1);
    }
}
```

Conclusion:

After exploring all these Object Streams, Java Swing, JPanel, and Java Swing thread topic, we learned a lot about painting transfer. Object Streams can represent many different kinds of sources and destinations, such as objects. Object Stream is a sequence of data. A program uses an InputStream to read data from a source, one item at a time; a program uses an OutputStream to write data to a destination, one item at time. In addition, JColorChooeser is a powerful tool in Java GUI to draw figures.