Name : Roger Ojar

ID : 809000518

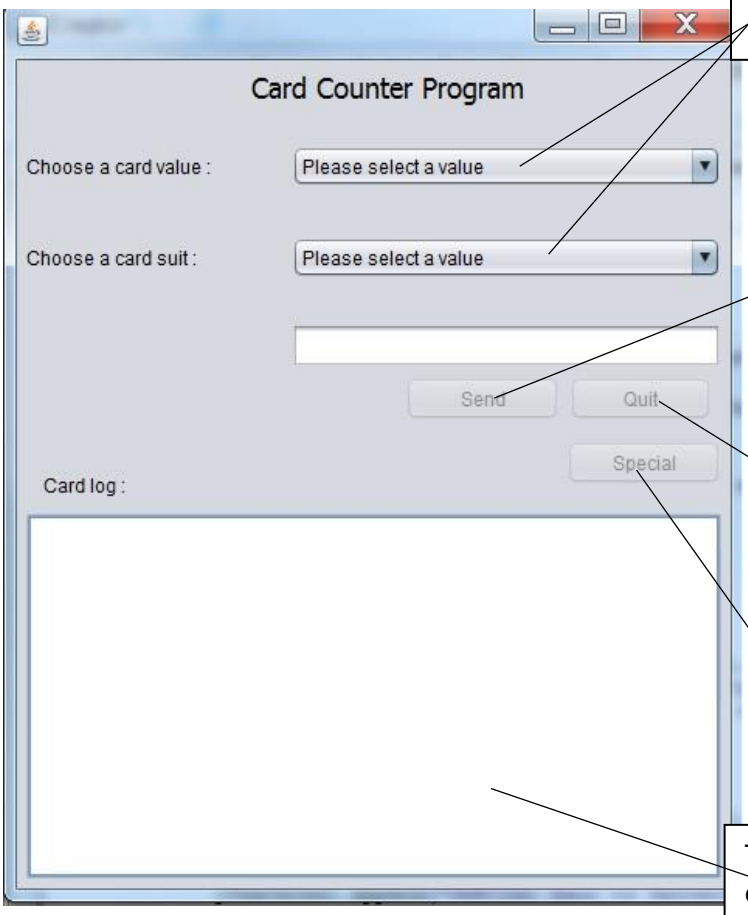
Assignment 1 - COMP 6601

Personal Assessment

My program works well and has very few crashes. The Server takes while to generate the card summary if there were many cards sent to the server due to having to draw each one, the special button takes a few moments occasionally to receive the reply form the server. I think I completed 100% of the requirements.

Explaining the GUI Elements

This is the Client Side UI.



There are two dropdowns to indicate which card you would like to send to the server.

The two dropdowns need to be used or the send button will not be enabled.

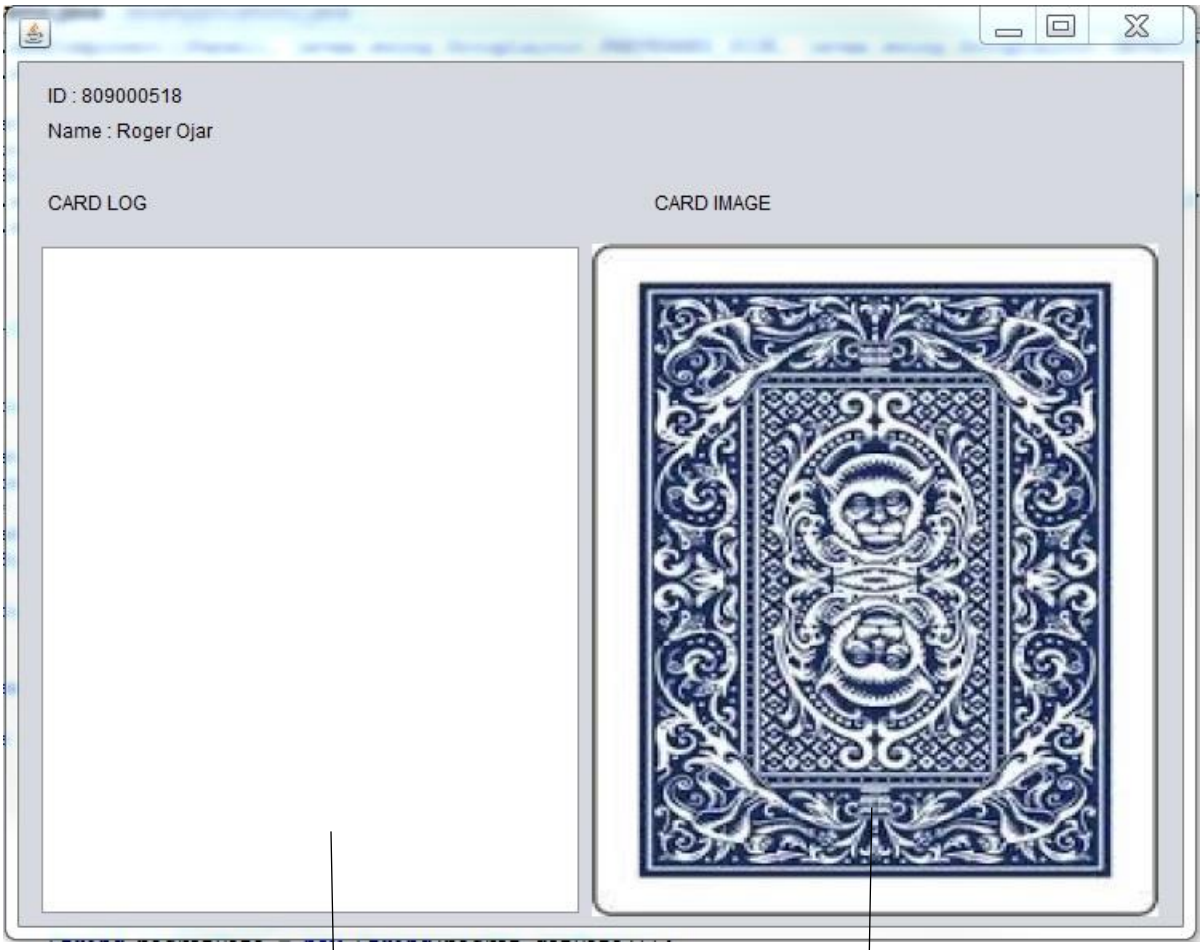
The Send button sends the card code in the text field which is populated by the dropdowns (eg. Queen of heart = QH)

The Quit button is enabled after sending your first message. It send the QUIT message to the server.

The Special button is enabled after five messages have been sent. This sends a message to the server to respond with how many card have been sent

The Card log Text area displays a log of what is being sent to the server.

This is the Server GUI



The Card log keeps track of messages incoming to the server.

The Card Image displays the card graphics of the incoming card.

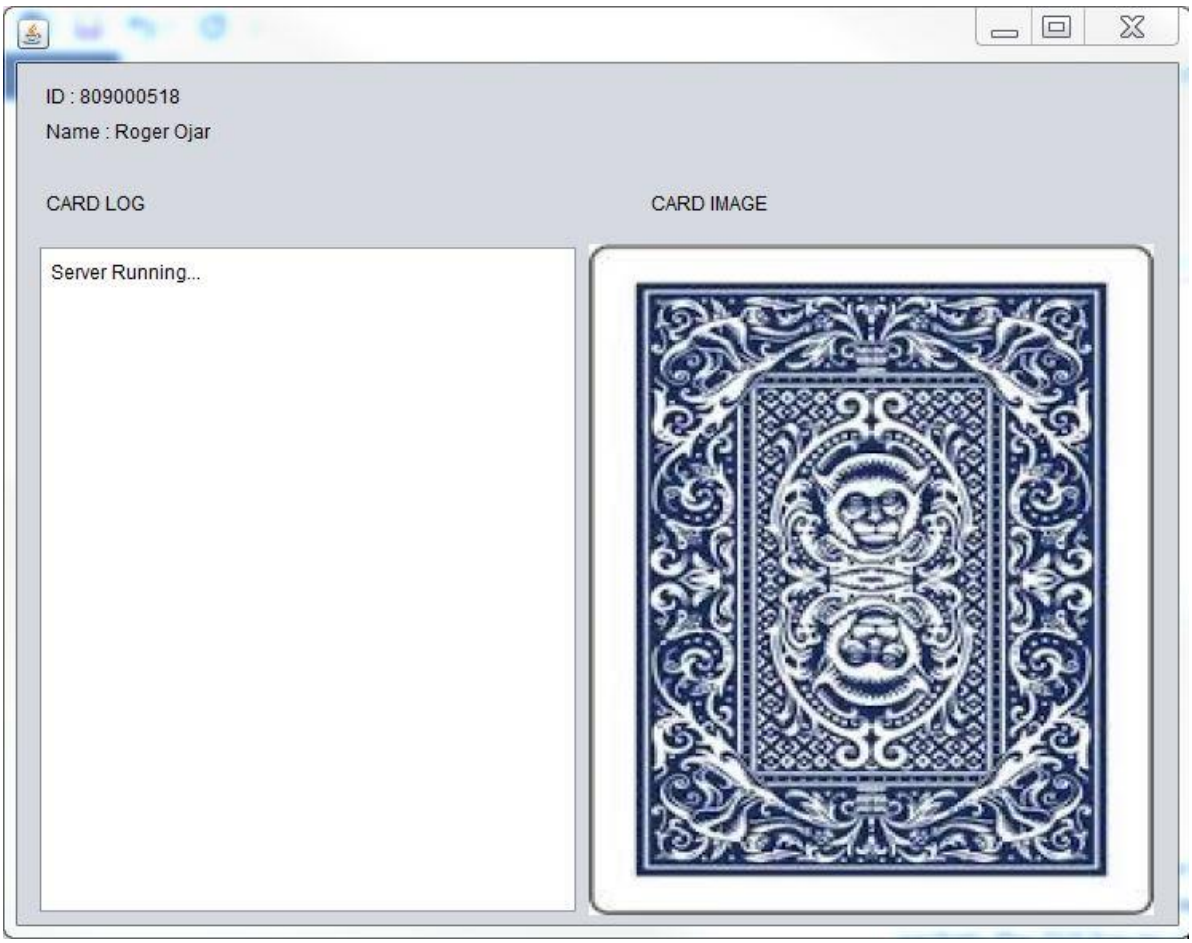
Sample Code and Execution of Program

Loading up the Server

```
public NewJFrame() {

displayPic();

      try{

         DatagramSocket aSocket = new DatagramSocket(6789);

         Connection c = new Connection(aSocket);

         Thread thread2 = new Thread(c);

         thread2.start();

        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());

           }

  }

class Connection extends Thread {
           DatagramSocket aSocket = null;
      public Connection (DatagramSocket aClientSocket) {
                        aSocket = aClientSocket;
           aSocketClient = aSocket;
           }
      public void run(){
         try{

                        jTextArea2.append("Server Running...\n");

         while (listen) {

              // receive request
              byte[] buf = new byte[10000];
              DatagramPacket packet = new DatagramPacket(buf, buf.length);
              aSocket.receive(packet);

              // figure out response
              String packetData = new String(packet.getData());
               senderPort = packet.getPort();

              senderAddress = packet.getAddress();
              displayPic2(packetData);
         }
         }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                        }catch (IOException e) {System.out.println("IO: " + e.getMessage());
                        }
           }
  }
```

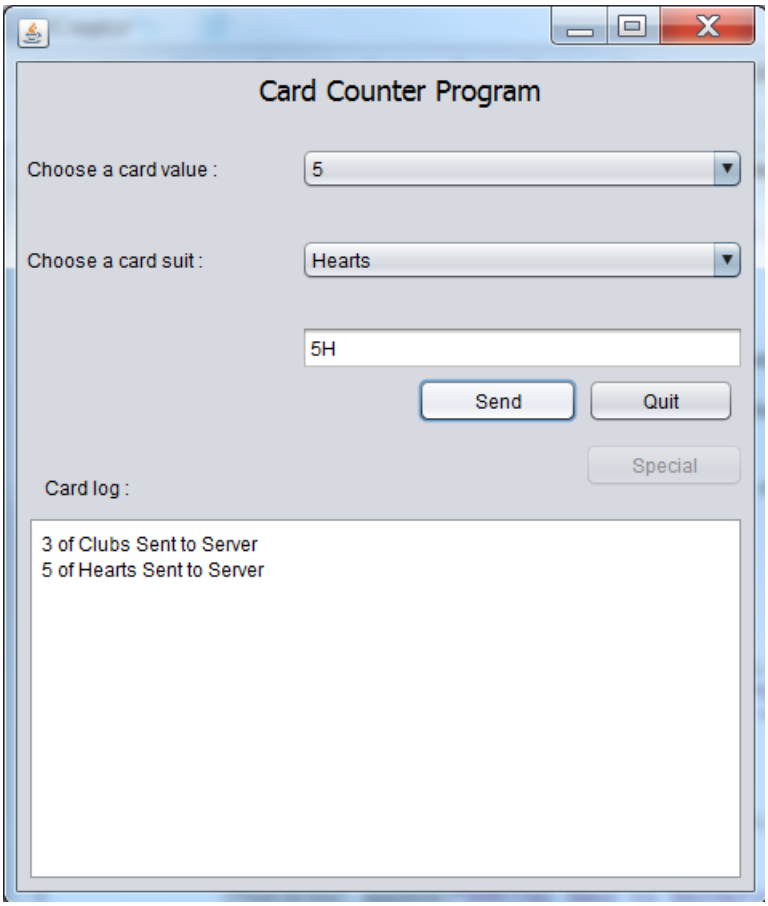
When the server launches this code will execute. This will initialize the UDP connection and start running it on a new thread. It needs to be threaded because while the server is waiting to receive packets the GUI has to update the card image and that cannot be done during the endless loop of receiving packets.

Server UI when launched :



Sending from client and receiving from server

The Server can now receive messages from the client.



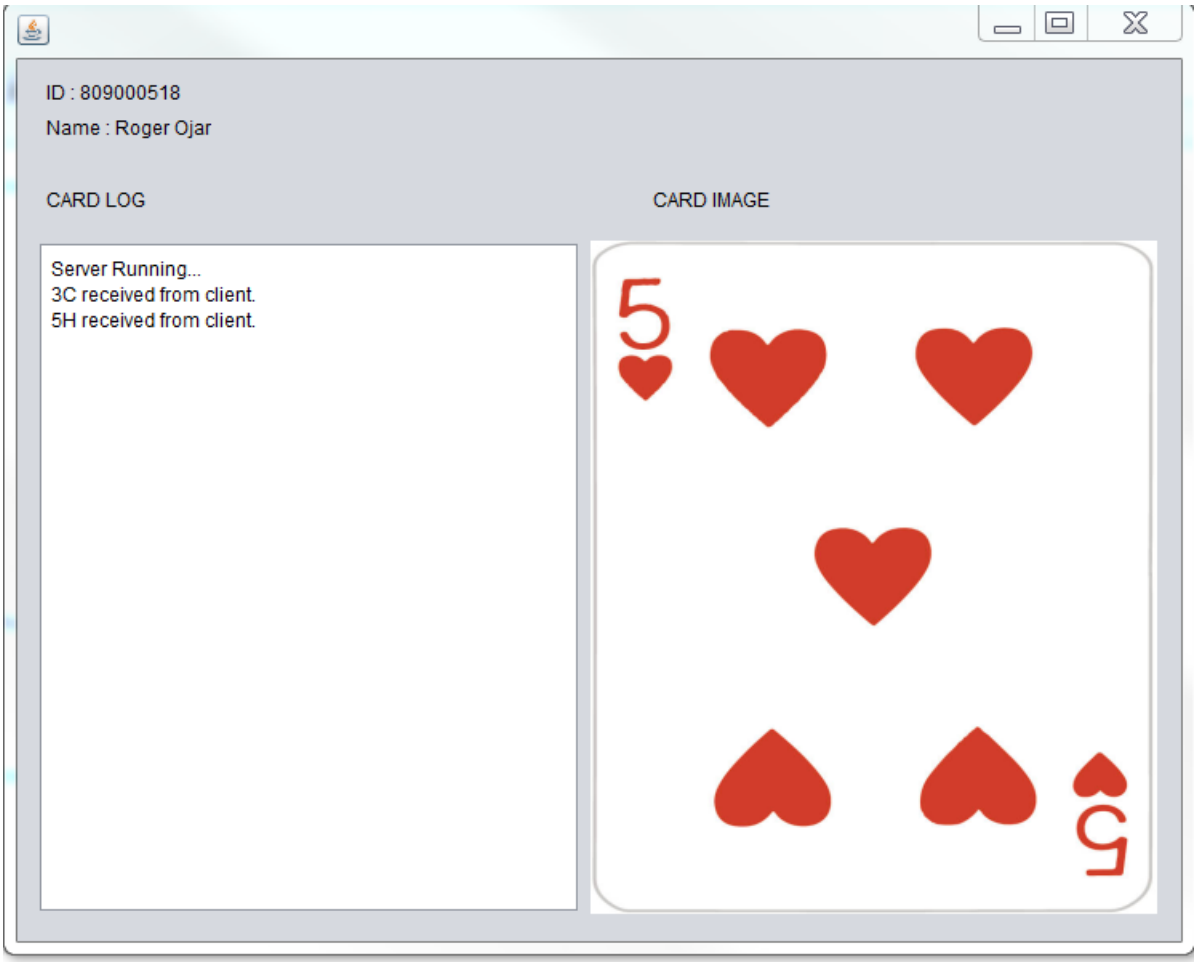
Here we see the client has selected the 5 of hearts and clicked Send.

```
try {
        byte [] m = null;
       aSocket = new DatagramSocket();
       InetAddress aHost = InetAddress.getByName(addressName);
        int serverPort = 6789;
         String s1 = String.valueOf(jTextField1.getText());
          m = s1.getBytes();
          DatagramPacket request =
          new DatagramPacket(m,  s1.length(), aHost, serverPort);
          aSocket.send(request);
           jButton3.setEnabled(true);
            countSpecial++;
           if(countSpecial >= 5){
          jButton2.setEnabled(true);
```

```
}
  jTextArea1.append(String.valueOf(jComboBox1.getSelectedItem()) + " of " +
 String.valueOf(jComboBox2.getSelectedItem()) + " Sent to Server\n");
            }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
            }catch (IOException e){System.out.println("IO: " + e.getMessage());
            }
```

When the Send button is clicked this code will execute and send the contents of the textbox to the server. It uses the address of the server by “addressname”, the “serverport”, converts the message to a byte array m and creates the datagramPacket to be sent to the server if the credentials are the same.

Below we can see the server has received the message and is displaying the card graphics.



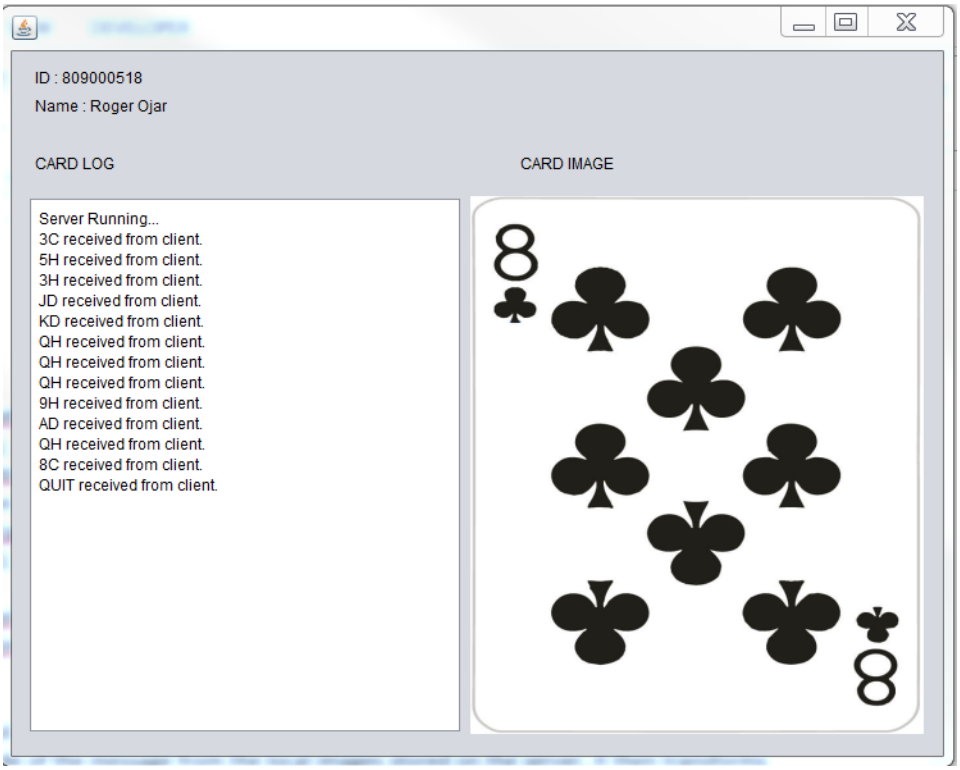
```
try{
            image = ImageIO.read(new File("JPEG_CARDS/" + str + ".jpg"));
            addCount(str);
            cardList.add(str);
        }catch (Exception e){
            e.printStackTrace();
            System.exit(1);
        }
        ImageIcon imageIcon = new ImageIcon(image.getScaledInstance(354, 420, 4));
        jLabel2.setIcon(imageIcon);
```

When a card is received and it is not a QUIT or SPECIAL message then the code executes to read the image based on the code of the message from the local images stored on the server. It then transforms a Jlabel stored in a Jframe to the image.

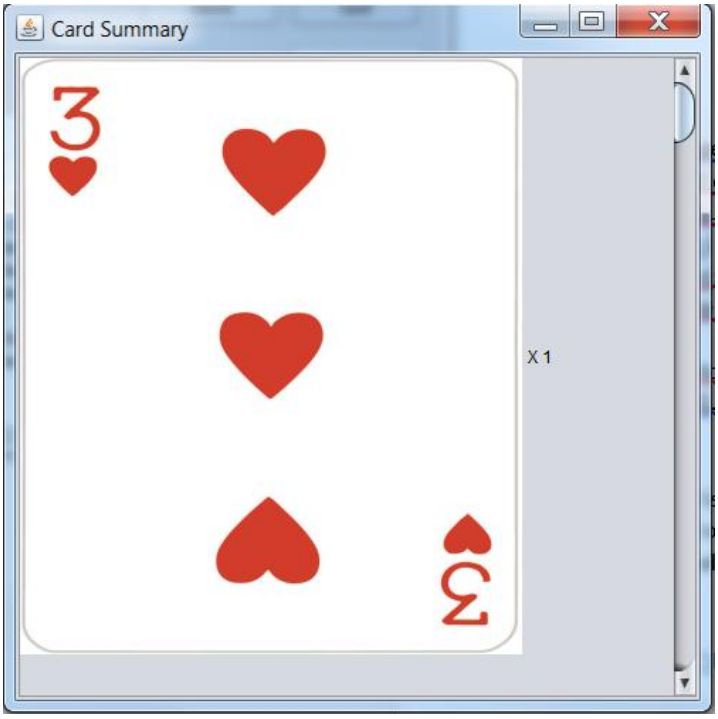
Sending a QUIT message

By Sending a QUIT message from the Client, the server is expected to show the totals of every card sent(including duplicates) with some form of graphics.

Below a QUIT message is received by the Server.



Another window will pop up with all the cards sent and a label next to them indicating the amount that was sent.



The card log shows four queens were sent and the result can be seen below with a “X 4” next to the queen of hearts.

```
JFrame editorFrame = new JFrame("Card Summary");
      JPanel jp = new JPanel();
      jp.setLayout(new BoxLayout(jp, BoxLayout.Y_AXIS));

      JLabel jLabel = new JLabel();
      for(int i = 0;i <= cardArr.length - 1; i++){

        if(cardCount[i] > 0){
                BufferedImage image = null;
                                try{
                                  image = ImageIO.read(new File("JPEG_CARDS/" + cardArr[i] + ".jpg"));
                                }catch (Exception e){
                                  e.printStackTrace();
                                  System.exit(1);
                                }
                  ImageIcon imageIcon2 = new ImageIcon(image.getScaledInstance(354, 420, 4));;
                  JLabel jLabeltemp = new JLabel("JLabel" + i);
                  jLabeltemp.setIcon(imageIcon2);

                  //name
                  jLabeltemp.setText("X " + cardCount[i]);

                  jp.add(jLabeltemp);
                 jp.add(Box.createRigidArea(new Dimension(0, 60)));
        }
      }

      jp.add(Box.createVerticalGlue());
      JScrollPane scrollPane = new JScrollPane(jp);
          editorFrame.setPreferredSize(new Dimension(500, 500));
      editorFrame.getContentPane().add(jp, BorderLayout.CENTER);
      editorFrame.add(new JScrollPane(jp));
      editorFrame.pack();
      editorFrame.setLocationRelativeTo(null);
      editorFrame.setVisible(true);
```

Two arrays are holding the cards and their counts. This code create a new Jframe, with a Jpanel in a box layout and iterates in the card array and stops on cards where the count is more than 0, which indicates a card was sent. That card is loaded and the count recorded, It then load a dynamically created Jlabel with the image and the count of the card.

Sending a SPECIAL message (extra feature)

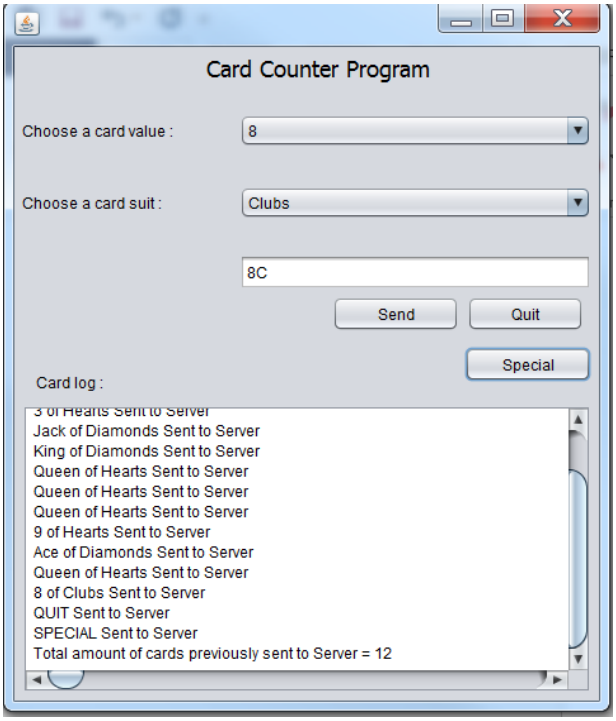
When the Special button is clicked, a SPECIAL message is sent to the server as well as a new thread is started to receive a replay form the Server.

```
new Thread() {
        public void run() {
          try {
                                if(spec == "SPECIAL"){
                                byte [] m = null;
                                aSocket = new DatagramSocket();
                                InetAddress aHost = InetAddress.getByName(addressName);
                                int serverPort = 6789;
                                      String s1 = spec;
                                           m = s1.getBytes();
                                DatagramPacket request =
                                new DatagramPacket(m,  s1.length(), aHost, serverPort);
                                aSocket.send(request);
                                byte[] buffer = new byte[100000];
                                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                                aSocket.receive(reply);
                                jTextArea1.append("Total amount of cards previously sent to Server = " + new
                                String(reply.getData()));
                                }
                                }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                                          }catch (IOException e){System.out.println("IO: " + e.getMessage());
                                          }catch (Exception e){
                                  e.printStackTrace();
                                  System.exit(1);
          }
         }
      }.start();
```

This needs to be threaded because the GUI has to be updated to present the information back the client.

```
if(cardList.size() >= 5){

          BufferedImage image = null;
         /*  String c1 = cardList.get(cardList.size() - 1);
          String c2 = cardList.get(cardList.size() - 2);
          String c3 = cardList.get(cardList.size() - 3);
          String c4 = cardList.get(cardList.size() - 4);
          String c5 = cardList.get(cardList.size() - 5);*/
                                int cardNum = 0;

          try{
             for(int i = 0;i <= cardArr.length - 1; i++){
                                if(cardCount[i] > 0){
                                        cardNum = cardNum + cardCount[i];
                              }
                             }
             String res = Integer.toString(cardNum);
             byte[] buffer = res.getBytes();
             DatagramPacket reply = new DatagramPacket(buffer, buffer.length,
senderAddress,senderPort);
             aSocketClient.send(reply);
            Thread.sleep(5000);
          }catch (Exception e){
              e.printStackTrace();
             System.exit(1);
          }
       }
```

This code on the server simply counts the amount of cards currently being stored on the card array, builds the datagramPacket and sends it back to the client. The Client address and port is stored from previous packets and reused here to build the Datagram Packet.



On the image above we can see the GUI being updated with the amount of cards sent to the server.