Source: http://www.dreamincode.net/forums/topic/259777-a-simple-chat-program-with-clientserver-gui-optional/

```
//-------------------------------------------------------------------------------------------
//  ChatMessage.java
//-------------------------------------------------------------------------------------------



import java.io.*;
/*
 * This class defines the different type of messages that will be exchanged between the
 * Clients and the Server.
 * When talking from a Java Client to a Java Server a lot easier to pass Java objects, no
 * need to count bytes or to wait for a line feed at the end of the frame
 */
public class ChatMessage implements Serializable {

    protected static final long serialVersionUID = 1112122200L;

    // The different types of message sent by the Client
    // WHOISIN to receive the list of the users connected
    // MESSAGE an ordinary message
    // LOGOUT to disconnect from the Server
    static final int WHOISIN = 0, MESSAGE = 1, LOGOUT = 2;
    private int type;
    private String message;

    // constructor
    ChatMessage(int type, String message) {
        this.type = type;
        this.message = message;
    }

    // getters
```

```
    int getType() {
        return type;
    }
    String getMessage() {
        return message;
    }
}
```

```
//-----------------------------------------------------------------------------------------
//  Server.java
//-----------------------------------------------------------------------------------------
```

```
import java.io.*;
import java.net.*;
import java.text.SimpleDateFormat;
import java.util.*;

/*
 * The server that can be run both as a console application or a GUI
 */
public class Server {
    // a unique ID for each connection
    private static int uniqueId;
    // an ArrayList to keep the list of the Client
    private ArrayList<ClientThread> al;
    // if I am in a GUI
    private ServerGUI sg;
    // to display time
    private SimpleDateFormat sdf;
    // the port number to listen for connection
    private int port;
    // the boolean that will be turned of to stop the server
    private boolean keepGoing;
```

```
/*
 *  server constructor that receive the port to listen to for connection as parameter
 *  in console
 */
public Server(int port) {
    this(port, null);
}

public Server(int port, ServerGUI sg) {
    // GUI or not
    this.sg = sg;
    // the port
    this.port = port;
    // to display hh:mm:ss
    sdf = new SimpleDateFormat("HH:mm:ss");
    // ArrayList for the Client list
    al = new ArrayList<ClientThread>();
}

public void start() {
    keepGoing = true;
    /* create socket server and wait for connection requests */
    try
    {
        // the socket used by the server
        ServerSocket serverSocket = new ServerSocket(port);

        // infinite loop to wait for connections
        while(keepGoing)
        {
            // format message saying we are waiting
            display("Server waiting for Clients on port " + port + ".");

            Socket socket = serverSocket.accept();      // accept connection
            // if I was asked to stop
            if(!keepGoing)
```

```
                    break;
                ClientThread t = new ClientThread(socket);  // make a thread of it
                al.add(t);                                  // save it in the ArrayList
                t.start();
            }
            // I was asked to stop
            try {
                serverSocket.close();
                for(int i = 0; i < al.size(); ++i) {
                    ClientThread tc = al.get(i);
                    try {
                    tc.sInput.close();
                    tc.sOutput.close();
                    tc.socket.close();
                    }
                    catch(IOException ioE) {
                        // not much I can do
                    }
                }
            }
            catch(Exception e) {
                display("Exception closing the server and clients: " + e);
            }
        }
        // something went bad
        catch (IOException e) {
            String msg = sdf.format(new Date()) + " Exception on new ServerSocket: " + e + "\n";
            display(msg);
        }
    }
    /*
     * For the GUI to stop the server
     */
    protected void stop() {
        keepGoing = false;
        // connect to myself as Client to exit statement
        // Socket socket = serverSocket.accept();
        try {
```

```
            new Socket("localhost", port);
        }
        catch(Exception e) {
            // nothing I can really do
        }
    }
    /*
     * Display an event (not a message) to the console or the GUI
     */
    private void display(String msg) {
        String time = sdf.format(new Date()) + " " + msg;
        if(sg == null)
            System.out.println(time);
        else
            sg.appendEvent(time + "\n");
    }
    /*
     *  to broadcast a message to all Clients
     */
    private synchronized void broadcast(String message) {
        // add HH:mm:ss and \n to the message
        String time = sdf.format(new Date());
        String messageLf = time + " " + message + "\n";
        // display message on console or GUI
        if(sg == null)
            System.out.print(messageLf);
        else
            sg.appendRoom(messageLf);     // append in the room window

        // we loop in reverse order in case we would have to remove a Client
        // because it has disconnected
        for(int i = al.size(); --i >= 0;) {
            ClientThread ct = al.get(i);
            // try to write to the Client if it fails remove it from the list
            if(!ct.writeMsg(messageLf)) {
                al.remove(i);
                display("Disconnected Client " + ct.username + " removed from list.");
            }
```

```
        }
    }

    // for a client who logoff using the LOGOUT message
    synchronized void remove(int id) {
        // scan the array list until we found the Id
        for(int i = 0; i < al.size(); ++i) {
            ClientThread ct = al.get(i);
            // found it
            if(ct.id == id) {
                al.remove(i);
                return;
            }
        }
    }

    /*
     *  To run as a console application just open a console window and:
     * > java Server
     * > java Server portNumber
     * If the port number is not specified 1500 is used
     */
    public static void main(String[] args) {
        // start server on port 1500 unless a PortNumber is specified
        int portNumber = 1500;
        switch(args.length) {
            case 1:
                try {
                    portNumber = Integer.parseInt(args[0]);
                }
                catch(Exception e) {
                    System.out.println("Invalid port number.");
                    System.out.println("Usage is: > java Server [portNumber]");
                    return;
                }
            case 0:
                break;
            default:
```

```
                System.out.println("Usage is: > java Server [portNumber]");
                return;

        }
        // create a server object and start it
        Server server = new Server(portNumber);
        server.start();
    }

    /** One instance of this thread will run for each client */
    class ClientThread extends Thread {
        // the socket where to listen/talk
        Socket socket;
        ObjectInputStream sInput;
        ObjectOutputStream sOutput;
        // my unique id (easier for deconnection)
        int id;
        // the Username of the Client
        String username;
        // the only type of message a will receive
        ChatMessage cm;
        // the date I connect
        String date;

        // Constructore
        ClientThread(Socket socket) {
            // a unique id
            id = ++uniqueId;
            this.socket = socket;
            /* Creating both Data Stream */
            System.out.println("Thread trying to create Object Input/Output Streams");
            try
            {
                // create output first
                sOutput = new ObjectOutputStream(socket.getOutputStream());
                sInput  = new ObjectInputStream(socket.getInputStream());
                // read the username
                username = (String) sInput.readObject();
```

```
            display(username + " just connected.");
        }
        catch (IOException e) {
            display("Exception creating new Input/output Streams: " + e);
            return;
        }
        // have to catch ClassNotFoundException
        // but I read a String, I am sure it will work
        catch (ClassNotFoundException e) {
        }
        date = new Date().toString() + "\n";
    }

    // what will run forever
    public void run() {
        // to loop until LOGOUT
        boolean keepGoing = true;
        while(keepGoing) {
            // read a String (which is an object)
            try {
                cm = (ChatMessage) sInput.readObject();
            }
            catch (IOException e) {
                display(username + " Exception reading Streams: " + e);
                break;
            }
            catch(ClassNotFoundException e2) {
                break;
            }
            // the messaage part of the ChatMessage
            String message = cm.getMessage();

            // Switch on the type of message receive
            switch(cm.getType()) {

            case ChatMessage.MESSAGE:
                broadcast(username + ": " + message);
                break;
```

```
            case ChatMessage.LOGOUT:
                display(username + " disconnected with a LOGOUT message.");
                keepGoing = false;
                break;
            case ChatMessage.WHOISIN:
                writeMsg("List of the users connected at " + sdf.format(new Date()) + "\n");
                // scan al the users connected
                for(int i = 0; i < al.size(); ++i) {
                    ClientThread ct = al.get(i);
                    writeMsg((i+1) + ") " + ct.username + " since " + ct.date);
                }
                break;
            }
        }
        // remove myself from the arrayList containing the list of the
        // connected Clients
        remove(id);
        close();
    }

    // try to close everything
    private void close() {
        // try to close the connection
        try {
            if(sOutput != null) sOutput.close();
        }
        catch(Exception e) {}
        try {
            if(sInput != null) sInput.close();
        }
        catch(Exception e) {};
        try {
            if(socket != null) socket.close();
        }
        catch (Exception e) {}
    }

    /*
```

```
         * Write a String to the Client output stream
         */
        private boolean writeMsg(String msg) {
            // if Client is still connected send the message to it
            if(!socket.isConnected()) {
                close();
                return false;
            }
            // write the message to the stream
            try {
                sOutput.writeObject(msg);
            }
            // if an error occurs, do not abort just inform the user
            catch(IOException e) {
                display("Error sending message to " + username);
                display(e.toString());
            }
            return true;
        }
    }
}




//-------------------------------------------------------------------------------------------
//  Client.java
//-------------------------------------------------------------------------------------------



import java.net.*;
import java.io.*;
import java.util.*;

/*
 * The Client that can be run both as a console or a GUI
 */
```

```
public class Client  {

    // for I/O
    private ObjectInputStream sInput;       // to read from the socket
    private ObjectOutputStream sOutput;     // to write on the socket
    private Socket socket;

    // if I use a GUI or not
    private ClientGUI cg;

    // the server, the port and the username
    private String server, username;
    private int port;

    /*
     *  Constructor called by console mode
     *  server: the server address
     *  port: the port number
     *  username: the username
     */
    Client(String server, int port, String username) {
        // which calls the common constructor with the GUI set to null
        this(server, port, username, null);
    }

    /*
     * Constructor call when used from a GUI
     * in console mode the ClienGUI parameter is null
     */
    Client(String server, int port, String username, ClientGUI cg) {
        this.server = server;
        this.port = port;
        this.username = username;
        // save if we are in GUI mode or not
        this.cg = cg;
    }

    /*
```

```
     * To start the dialog
     */
    public boolean start() {
        // try to connect to the server
        try {
            socket = new Socket(server, port);
        }
        // if it failed not much I can so
        catch(Exception ec) {
            display("Error connectiong to server:" + ec);
            return false;
        }

        String msg = "Connection accepted " + socket.getInetAddress() + ":" + socket.getPort();
        display(msg);

        /* Creating both Data Stream */
        try
        {
            sInput  = new ObjectInputStream(socket.getInputStream());
            sOutput = new ObjectOutputStream(socket.getOutputStream());
        }
        catch (IOException eIO) {
            display("Exception creating new Input/output Streams: " + eIO);
            return false;
        }

        // creates the Thread to listen from the server
        new ListenFromServer().start();
        // Send our username to the server this is the only message that we
        // will send as a String. All other messages will be ChatMessage objects
        try
        {
            sOutput.writeObject(username);
        }
        catch (IOException eIO) {
            display("Exception doing login : " + eIO);
            disconnect();
```

```
            return false;
        }
        // success we inform the caller that it worked
        return true;
    }

    /*
     * To send a message to the console or the GUI
     */
    private void display(String msg) {
        if(cg == null)
            System.out.println(msg);      // println in console mode
        else
            cg.append(msg + "\n");      // append to the ClientGUI JTextArea (or whatever)
    }

    /*
     * To send a message to the server
     */
    void sendMessage(ChatMessage msg) {
        try {
            sOutput.writeObject(msg);
        }
        catch(IOException e) {
            display("Exception writing to server: " + e);
        }
    }

    /*
     * When something goes wrong
     * Close the Input/Output streams and disconnect not much to do in the catch clause
     */
    private void disconnect() {
        try {
            if(sInput != null) sInput.close();
        }
        catch(Exception e) {} // not much else I can do
        try {
```

```
            if(sOutput != null) sOutput.close();
        }
        catch(Exception e) {} // not much else I can do
        try{
            if(socket != null) socket.close();
        }
        catch(Exception e) {} // not much else I can do

        // inform the GUI
        if(cg != null)
            cg.connectionFailed();

    }
    /*
     * To start the Client in console mode use one of the following command
     * > java Client
     * > java Client username
     * > java Client username portNumber
     * > java Client username portNumber serverAddress
     * at the console prompt
     * If the portNumber is not specified 1500 is used
     * If the serverAddress is not specified "localHost" is used
     * If the username is not specified "Anonymous" is used
     * > java Client
     * is equivalent to
     * > java Client Anonymous 1500 localhost
     * are eqquivalent
     *
     * In console mode, if an error occurs the program simply stops
     * when a GUI id used, the GUI is informed of the disconnection
     */
    public static void main(String[] args) {
        // default values
        int portNumber = 1500;
        String serverAddress = "localhost";
        String userName = "Anonymous";

        // depending of the number of arguments provided we fall through
```

```
        switch(args.length) {
            // > javac Client username portNumber serverAddr
            case 3:
                serverAddress = args[2];
            // > javac Client username portNumber
            case 2:
                try {
                    portNumber = Integer.parseInt(args[1]);
                }
                catch(Exception e) {
                    System.out.println("Invalid port number.");
                    System.out.println("Usage is: > java Client [username] [portNumber] [serverAddress]");
                    return;
                }
            // > javac Client username
            case 1:
                userName = args[0];
            // > java Client
            case 0:
                break;
            // invalid number of arguments
            default:
                System.out.println("Usage is: > java Client [username] [portNumber] {serverAddress]");
            return;
        }
        // create the Client object
        Client client = new Client(serverAddress, portNumber, userName);
        // test if we can start the connection to the Server
        // if it failed nothing we can do
        if(!client.start())
            return;

        // wait for messages from user
        Scanner scan = new Scanner(System.in);
        // loop forever for message from the user
        while(true) {
            System.out.print("> ");
            // read message from user
```

```
            String msg = scan.nextLine();
            // logout if message is LOGOUT
            if(msg.equalsIgnoreCase("LOGOUT")) {
                client.sendMessage(new ChatMessage(ChatMessage.LOGOUT, ""));
                // break to do the disconnect
                break;
            }
            // message WhoIsIn
            else if(msg.equalsIgnoreCase("WHOISIN")) {
                client.sendMessage(new ChatMessage(ChatMessage.WHOISIN, ""));
            }
            else {              // default to ordinary message
                client.sendMessage(new ChatMessage(ChatMessage.MESSAGE, msg));
            }
        }
        // done disconnect
        client.disconnect();
    }

    /*
     * a class that waits for the message from the server and append them to the JTextArea
     * if we have a GUI or simply System.out.println() it in console mode
     */
    class ListenFromServer extends Thread {

        public void run() {
            while(true) {
                try {
                    String msg = (String) sInput.readObject();
                    // if console mode print the message and add back the prompt
                    if(cg == null) {
                        System.out.println(msg);
                        System.out.print("> ");
                    }
                    else {
                        cg.append(msg);
                    }
                }
```

```
                catch(IOException e) {
                    display("Server has close the connection: " + e);
                    if(cg != null)
                        cg.connectionFailed();
                    break;
                }
                // can't happen with a String object but need the catch anyhow
                catch(ClassNotFoundException e2) {
                }
            }
        }
    }
}




//--------------------------------------------------------------------------------------------
//  ServerGUI.Java
//--------------------------------------------------------------------------------------------



import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/*
 * The server as a GUI
 */
public class ServerGUI extends JFrame implements ActionListener, WindowListener {

    private static final long serialVersionUID = 1L;
    // the stop and start buttons
    private JButton stopStart;
    // JTextArea for the chat room and the events
    private JTextArea chat, event;
    // The port number
```

```
    private JTextField tPortNumber;
    // my server
    private Server server;


    // server constructor that receive the port to listen to for connection as parameter
    ServerGUI(int port) {
        super("Chat Server");
        server = null;
        // in the NorthPanel the PortNumber the Start and Stop buttons
        JPanel north = new JPanel();
        north.add(new JLabel("Port number: "));
        tPortNumber = new JTextField("  " + port);
        north.add(tPortNumber);
        // to stop or start the server, we start with "Start"
        stopStart = new JButton("Start");
        stopStart.addActionListener(this);
        north.add(stopStart);
        add(north, BorderLayout.NORTH);

        // the event and chat room
        JPanel center = new JPanel(new GridLayout(2,1));
        chat = new JTextArea(80,80);
        chat.setEditable(false);
        appendRoom("Chat room.\n");
        center.add(new JScrollPane(chat));
        event = new JTextArea(80,80);
        event.setEditable(false);
        appendEvent("Events log.\n");
        center.add(new JScrollPane(event));
        add(center);

        // need to be informed when the user click the close button on the frame
        addWindowListener(this);
        setSize(400, 600);
        setVisible(true);
    }
```

```
    // append message to the two JTextArea
    // position at the end
    void appendRoom(String str) {
        chat.append(str);
        chat.setCaretPosition(chat.getText().length() - 1);
    }
    void appendEvent(String str) {
        event.append(str);
        event.setCaretPosition(chat.getText().length() - 1);

    }

    // start or stop where clicked
    public void actionPerformed(ActionEvent e) {
        // if running we have to stop
        if(server != null) {
            server.stop();
            server = null;
            tPortNumber.setEditable(true);
            stopStart.setText("Start");
            return;
        }
        // OK start the server
        int port;
        try {
            port = Integer.parseInt(tPortNumber.getText().trim());
        }
        catch(Exception er) {
            appendEvent("Invalid port number");
            return;
        }
        // ceate a new Server
        server = new Server(port, this);
        // and start it as a thread
        new ServerRunning().start();
        stopStart.setText("Stop");
        tPortNumber.setEditable(false);
    }
```

```
// entry point to start the Server
public static void main(String[] arg) {
    // start server default port 1500
    new ServerGUI(1500);
}

/*
 * If the user click the X button to close the application
 * I need to close the connection with the server to free the port
 */
public void windowClosing(WindowEvent e) {
    // if my Server exist
    if(server != null) {
        try {
            server.stop();          // ask the server to close the conection
        }
        catch(Exception eClose) {
        }
        server = null;
    }
    // dispose the frame
    dispose();
    System.exit(0);
}
// I can ignore the other WindowListener method
public void windowClosed(WindowEvent e) {}
public void windowOpened(WindowEvent e) {}
public void windowIconified(WindowEvent e) {}
public void windowDeiconified(WindowEvent e) {}
public void windowActivated(WindowEvent e) {}
public void windowDeactivated(WindowEvent e) {}

/*
 * A thread to run the Server
 */
class ServerRunning extends Thread {
    public void run() {
```

```
            server.start();         // should execute until if fails
            // the server failed
            stopStart.setText("Start");
            tPortNumber.setEditable(true);
            appendEvent("Server crashed\n");
            server = null;
        }
    }

}




//-----------------------------------------------------------------------------------------
//  ClientGUI.java
//-----------------------------------------------------------------------------------------



import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


/*
 * The Client with its GUI
 */
public class ClientGUI extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;
    // will first hold "Username:", later on "Enter message"
    private JLabel label;
    // to hold the Username and later on the messages
    private JTextField tf;
    // to hold the server address an the port number
    private JTextField tfServer, tfPort;
    // to Logout and get the list of the users
```

```
    private JButton login, logout, whoIsIn;
    // for the chat room
    private JTextArea ta;
    // if it is for connection
    private boolean connected;
    // the Client object
    private Client client;
    // the default port number
    private int defaultPort;
    private String defaultHost;

    // Constructor connection receiving a socket number
    ClientGUI(String host, int port) {

        super("Chat Client");
        defaultPort = port;
        defaultHost = host;

        // The NorthPanel with:
        JPanel northPanel = new JPanel(new GridLayout(3,1));
        // the server name anmd the port number
        JPanel serverAndPort = new JPanel(new GridLayout(1,5, 1, 3));
        // the two JTextField with default value for server address and port number
        tfServer = new JTextField(host);
        tfPort = new JTextField("" + port);
        tfPort.setHorizontalAlignment(SwingConstants.RIGHT);

        serverAndPort.add(new JLabel("Server Address:  "));
        serverAndPort.add(tfServer);
        serverAndPort.add(new JLabel("Port Number:  "));
        serverAndPort.add(tfPort);
        serverAndPort.add(new JLabel(""));
        // adds the Server an port field to the GUI
        northPanel.add(serverAndPort);

        // the Label and the TextField
        label = new JLabel("Enter your username below", SwingConstants.CENTER);
        northPanel.add(label);
```

```
        tf = new JTextField("Anonymous");
        tf.setBackground(Color.WHITE);
        northPanel.add(tf);
        add(northPanel, BorderLayout.NORTH);

        // The CenterPanel which is the chat room
        ta = new JTextArea("Welcome to the Chat room\n", 80, 80);
        JPanel centerPanel = new JPanel(new GridLayout(1,1));
        centerPanel.add(new JScrollPane(ta));
        ta.setEditable(false);
        add(centerPanel, BorderLayout.CENTER);

        // the 3 buttons
        login = new JButton("Login");
        login.addActionListener(this);
        logout = new JButton("Logout");
        logout.addActionListener(this);
        logout.setEnabled(false);       // you have to login before being able to logout
        whoIsIn = new JButton("Who is in");
        whoIsIn.addActionListener(this);
        whoIsIn.setEnabled(false);      // you have to login before being able to Who is in

        JPanel southPanel = new JPanel();
        southPanel.add(login);
        southPanel.add(logout);
        southPanel.add(whoIsIn);
        add(southPanel, BorderLayout.SOUTH);

        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(600, 600);
        setVisible(true);
        tf.requestFocus();

    }

    // called by the Client to append text in the TextArea
    void append(String str) {
        ta.append(str);
```

```
        ta.setCaretPosition(ta.getText().length() - 1);
    }
    // called by the GUI is the connection failed
    // we reset our buttons, label, textfield
    void connectionFailed() {
        login.setEnabled(true);
        logout.setEnabled(false);
        whoIsIn.setEnabled(false);
        label.setText("Enter your username below");
        tf.setText("Anonymous");
        // reset port number and host name as a construction time
        tfPort.setText("" + defaultPort);
        tfServer.setText(defaultHost);
        // let the user change them
        tfServer.setEditable(false);
        tfPort.setEditable(false);
        // don't react to a <CR> after the username
        tf.removeActionListener(this);
        connected = false;
    }

    /*
    * Button or JTextField clicked
    */
    public void actionPerformed(ActionEvent e) {
        Object o = e.getSource();
        // if it is the Logout button
        if(o == logout) {
            client.sendMessage(new ChatMessage(ChatMessage.LOGOUT, ""));
            return;
        }
        // if it the who is in button
        if(o == whoIsIn) {
            client.sendMessage(new ChatMessage(ChatMessage.WHOISIN, ""));
            return;
        }

        // ok it is coming from the JTextField
```

```
        if(connected) {
            // just have to send the message
            client.sendMessage(new ChatMessage(ChatMessage.MESSAGE, tf.getText()));
            tf.setText("");
            return;
        }


        if(o == login) {
            // ok it is a connection request
            String username = tf.getText().trim();
            // empty username ignore it
            if(username.length() == 0)
                return;
            // empty serverAddress ignore it
            String server = tfServer.getText().trim();
            if(server.length() == 0)
                return;
            // empty or invalid port numer, ignore it
            String portNumber = tfPort.getText().trim();
            if(portNumber.length() == 0)
                return;
            int port = 0;
            try {
                port = Integer.parseInt(portNumber);
            }
            catch(Exception en) {
                return;   // nothing I can do if port number is not valid
            }

            // try creating a new Client with GUI
            client = new Client(server, port, username, this);
            // test if we can start the Client
            if(!client.start())
                return;
            tf.setText("");
            label.setText("Enter your message below");
            connected = true;
```

```
            // disable login button
            login.setEnabled(false);
            // enable the 2 buttons
            logout.setEnabled(true);
            whoIsIn.setEnabled(true);
            // disable the Server and Port JTextField
            tfServer.setEditable(false);
            tfPort.setEditable(false);
            // Action listener for when the user enter a message
            tf.addActionListener(this);
        }

    }

    // to start the whole thing the server
    public static void main(String[] args) {
        new ClientGUI("localhost", 1500);
    }

}
```