Projektrapport

Chattapplikation
för Objektorienterad programutveckling, trådar och datakommunikation

Rasmus Andersson
Emil Sandgren
Erik Sandgren
Jimmy Maksymiw
Lorenz Puskas
Kalle Bornemark

13 mars 2015

Innehåll

1 Arbetsbeskrivning

1.1 Rasmus Andersson

Arbetade med kommunikation mellan servern och klienten med Kalle Bornemark, och Jimmy Maksymiw. Formgav projektrapporten samt skrev ImageScaleHandler.java samt Chatlog.java. Jobbade inte med UI-klasserna.

1.2 Emil Sandgren

1.3 Erik Sandgren

Arbetat med generell grundläggande kommunikation mellan server och klient i början. Jobbat sedan med UI och hoppat in lite därefter på det som behövdes. Har ritat upp strukturen mycket och buggfixat.

1.4 Jimmy Maksymiw

1.5 Lorenz Puskas

Arbetade enbart med att designa ClientUI tillsammans med Emil.

1.6 Kalle Bornemark

2 Instruktioner för programstart

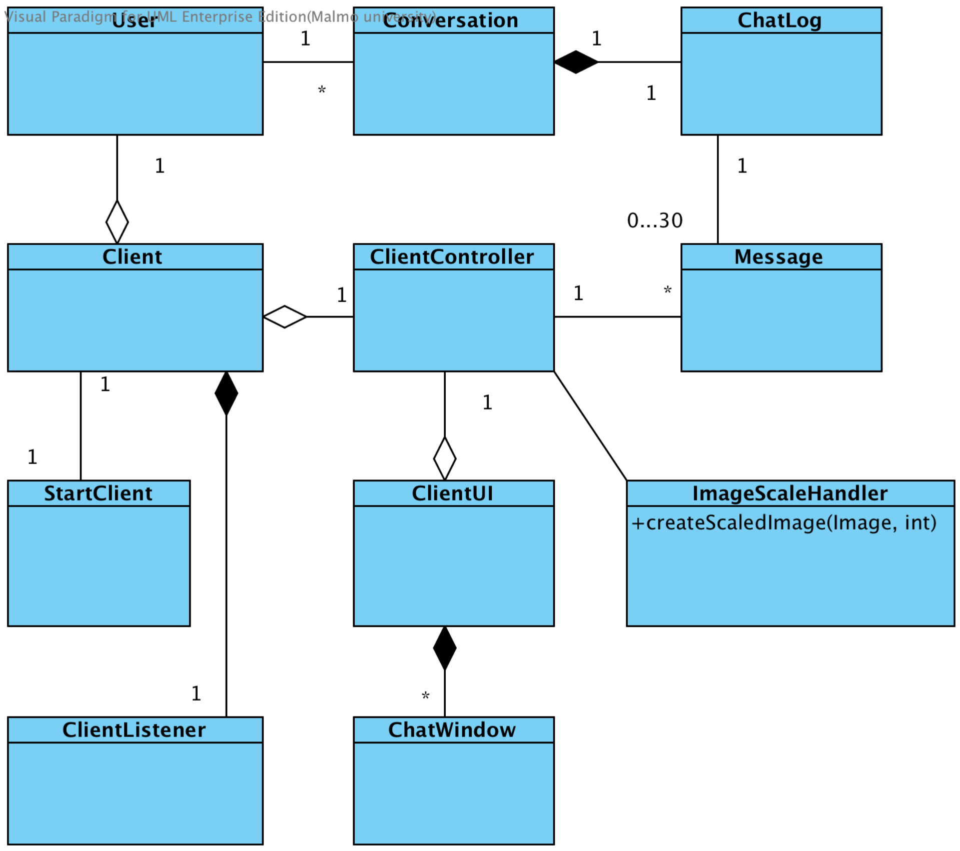
För att köra programmet så krävs det att man startar en server och minst en klient. Main-metoden för att starta servern finns i StartServer.java och main-metoden för att starta klienter finns i StartClient.java. Alla filvägar är relativa till det workspace som används och behöver inte ändras.

3 Systembeskrivning

Vårt system förser en Chatt-tjänst. I systemet finns det klienter och en server. Klienterna har ett grafiskt användargränssnitt som han eller hon kan använda för att skicka meddelanden till alla andra anslutna klienter, enskilda klienter, eller till en grupp av klienter. Meddelanden består av text eller av bilder. Alla dessa meddelanden går via en server som ser till att meddelanden kommer fram till rätt gruppchat eller till lobbyn. Servern lagrar alla textmeddelande som användarna skickar och loggar även namnet på de bilder som skickas via bildmeddelanden. Det loggas även när användare ansluter eller stänger ner anslutningen mot servern.

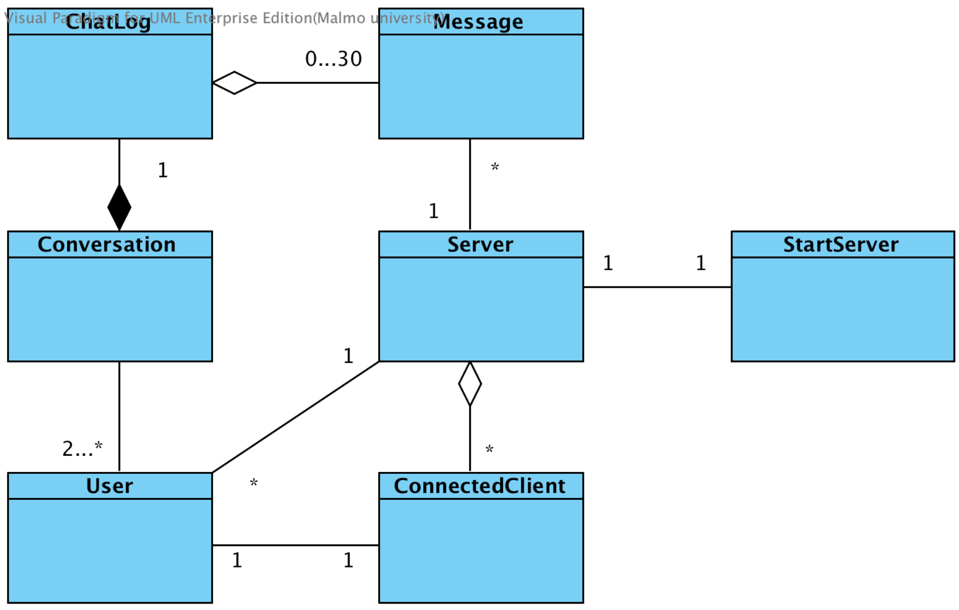
4 Klassdiagram

4.1 Server



Figur 1: Server

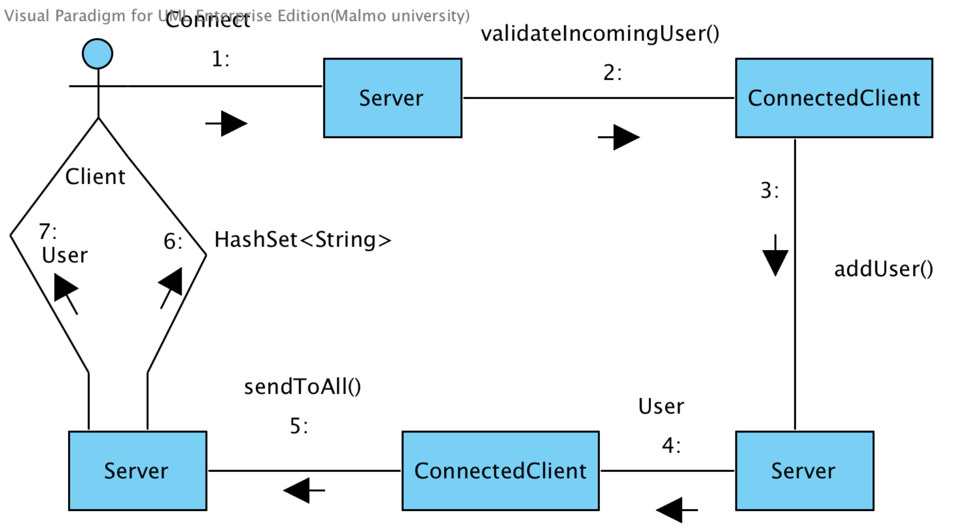
4.2 Klient



Figur 2: Klient

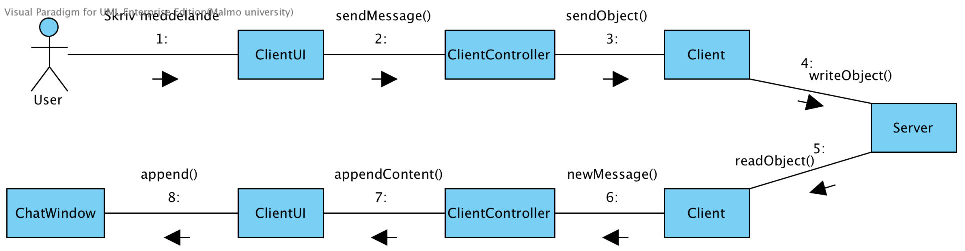
5 Kommunikationsdiagram

5.1 Connect and login



Figur 3: Client connecting and logging in

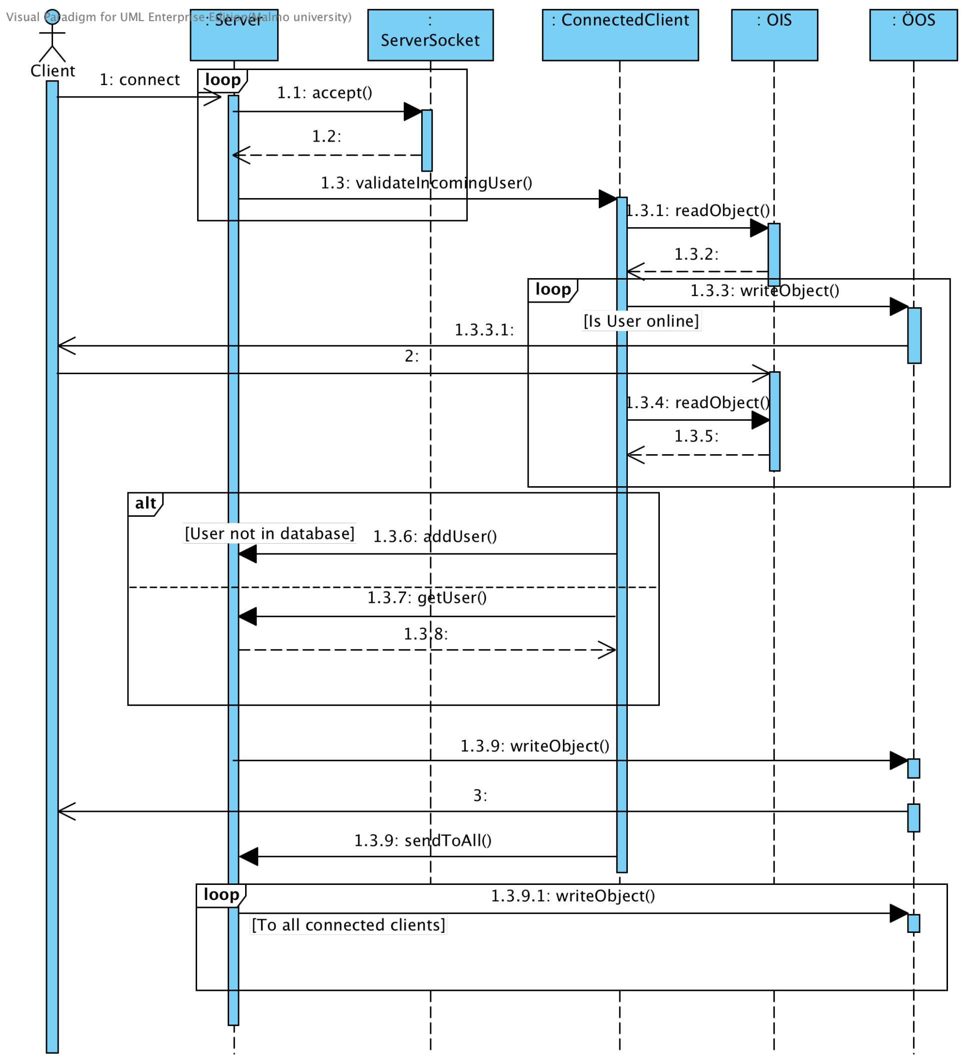
5.2 Client send Message



Figur 4: Client sending a message

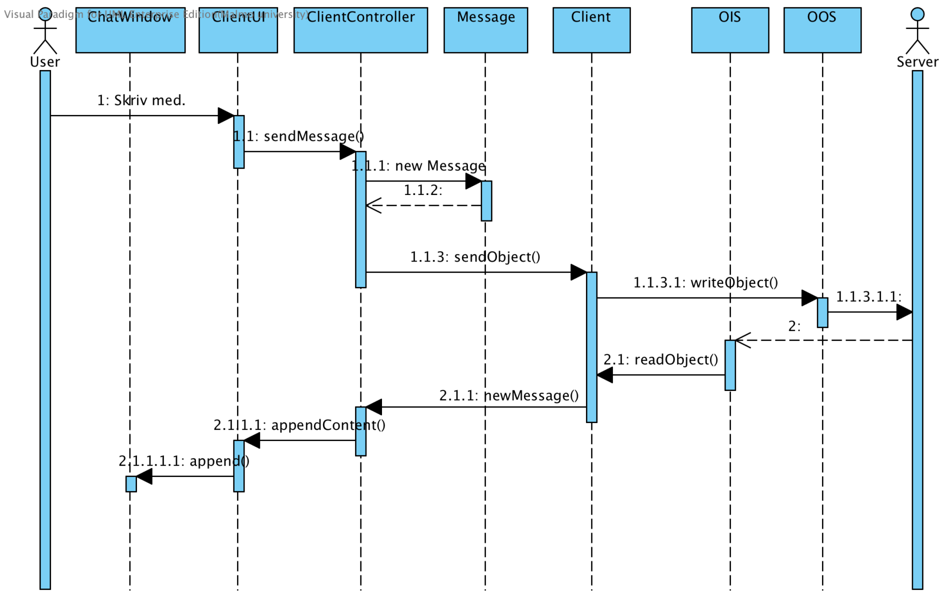
6 Sekvensdiagram

6.1 Connect and login



Figur 5: Client connecting and logging in

6.2 Send message



Figur 6: Client sending a message

7 Källkod

7.1 Server

7.1.1 Server.java, Server.ConnectedClient.java

```java
package chat;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.logging.*;

/**
 * Model class for the server.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class Server implements Runnable {
    private ServerSocket serverSocket;
    private ArrayList<ConnectedClient> connectedClients;
    private ArrayList<User> registeredUsers;
    private static final Logger LOGGER = Logger.getLogger(Server
        .class.getName());

    public Server(int port) {
        initLogger();
        registeredUsers = new ArrayList<>();
        connectedClients = new ArrayList<>();
        try {
            serverSocket = new ServerSocket(port);
            new Thread(this).start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Initiates the Logger
     */
    private void initLogger() {
        Handler fh;
        try {
            fh = new FileHandler("./src/log/Server.log");
            LOGGER.addHandler(fh);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);
            LOGGER.setLevel(Level.FINE);
        } catch (IOException e) {}
    }

    /**
     * Returns the User which ID matches the given ID.
     * Returns null if it doesn't exist.
     *
     * @param id The ID of the User that is to be found.
     * @return The matching User object, or null.
     */
    public User getUser(String id) {
        for (User user : registeredUsers) {
            if (user.getId().equals(id)) {
                return user;
            }
        }
        return null;
    }

    /**
     * Sends an object to all currently connected clients.
     *
     * @param object The object to be sent.
     */
    public synchronized void sendObjectToAll(Object object) {
        for (ConnectedClient client : connectedClients) {
            client.sendObject(object);
        }
    }

    /**
     * Checks who the message shall be sent to, then sends it.
     *
     * @param message The message to be sent.
     */
    public void sendMessage(Message message) {
        Conversation conversation = null;
        String to = "";

        // Lobby message
        if (message.getConversationID() == -1) {
            sendObjectToAll(message);
            to += "lobby";
        } else {
            User senderUser = null;

            // Finds the sender user
            for (ConnectedClient cClient : connectedClients) {
                if (cClient.getUser().getId().equals(message.getFromUserID())) {
                    senderUser = cClient.getUser();

                    // Finds the conversation the message shall be sent to
                    for (Conversation con : senderUser.getConversations()) {
                        if (con.getId() == message.getConversationID()) {
                            conversation = con;
                            to += conversation.getInvolvedUsers().toString();

                            // Finds the message's recipient users, then sends the message
                            for (String s : con.getInvolvedUsers()) {
                                for (ConnectedClient conClient : connectedClients) {
                                    if (conClient.getUser().getId().equals(s)) {
                                        conClient.sendObject(message);
                                    }
                                }
                            }
                            conversation.addMessage(message);
                        }
                    }
                }
            }
        }
        LOGGER.info("-- NEW MESSAGE SENT --\n" +
                "From: " + message.getFromUserID() + "\n" +
                "To: " + to + "\n" +
                "Message: " + message.getContent().toString());
    }

    /**
     * Sends a Conversation object to its involved users
     *
     * @param conversation The Conversation object to be sent.
     */
    public void sendConversation(Conversation conversation) {
        HashSet<String> users = conversation.getInvolvedUsers();
        for (String s : users) {
            for (ConnectedClient c : connectedClients) {
                if (c.getUser().getId().equals(s)) {
                    c.sendObject(conversation);
                }
            }
        }
    }

    /**
     * Sends an ArrayList with all connected user's IDs.
     */
    public void sendConnectedClients() {
        ArrayList<String> connectedUsers = new ArrayList<>();
        for (ConnectedClient client : connectedClients) {
            connectedUsers.add(client.getUser().getId());
        }
        sendObjectToAll(connectedUsers);
    }

    /**
     * Waits for client to connect.
     * Creates a new instance of ConnectedClient upon client
       connection.
     * Adds client to list of connected clients.
     */
    public void run() {
        LOGGER.info("Server started.");
        while (true) {
            try {
                Socket socket = serverSocket.accept();
                ConnectedClient client = new ConnectedClient(
                    socket, this);
                connectedClients.add(client);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
```

```
     * Class to handle the communication between server and connected clients.
     */
    private class ConnectedClient implements Runnable {
        private Thread client = new Thread(this);
        private ObjectOutputStream oos;
        private ObjectInputStream ois;
        private Server server;
        private User user;
        private Socket socket;

        public ConnectedClient(Socket socket, Server server) {
            LOGGER.info("Client connected: " + socket.getInetAddress());
            this.socket = socket;
            this.server = server;
            try {
                oos = new ObjectOutputStream(socket.getOutputStream());
                ois = new ObjectInputStream(socket.getInputStream());
            } catch (IOException e) {
                e.printStackTrace();
            }
            client.start();
        }

        /**
         * Returns the connected clients current User.
         *
         * @return The connected clients current User
         */
        public User getUser() {
            return user;
        }

        /**
         * Sends an object to the client.
         *
         * @param object The object to be sent.
         */
        public synchronized void sendObject(Object object) {
            try {
                oos.writeObject(object);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        /**
         * Removes the user from the list of connected clients.
         */
        public void removeConnectedClient() {
            for (int i = 0; i < connectedClients.size(); i++) {
```

```java
                if (connectedClients.get(i).getUser().getId().
                    equals(this.getUser().getId())) {
                    connectedClients.remove(i);
                    System.out.println("Client removed from
                        connectedClients");
                }
            }
        }

        /**
         * Removes the connected client,
         * sends an updated list of connected clients to other
            connected clients,
         * sends a server message with information of who
            disconnected
         * and closes the client's socket.
         */
        public void disconnectClient() {
            removeConnectedClient();
            sendConnectedClients();
            sendObjectToAll("Client disconnected: " + user.getId
                ());
            LOGGER.info("Client disconnected: " + user.getId());
            try {
                socket.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        /**
         * Checks if given user exists among already registered
            users.
         *
         * @return Whether given user already exists or not.
         */
        public boolean isUserInDatabase(User user) {
            for (User u : registeredUsers) {
                if (u.getId().equals(user.getId())) {
                    return true;
                }
            }
            return false;
        }

        public User getUser(String ID) {
            for (User user : registeredUsers) {
                if (user.getId().equals(ID)) {
                    return user;
                }
            }
            return null;
        }

        /**
         * Compare given user ID with connected client's IDs and
             check if the user is online.
         *
         * @param id User ID to check online status.
         * @return Whether given user is online or not.
         */
        public boolean isUserOnline(String id) {
            for (ConnectedClient client : connectedClients) {

                if (client.getUser().getId().equals(id) &&
                    client != this) {
                    return true;
                }
            }
            return false;
        }

        /**
         * Checks if given set of User IDs already has an open
           conversation.
         * If it does, it sends the conversation to its
           participants.
         * If it doesn't, it creates a new conversation, adds it
             to the current users
         * conversation list, and sends the conversation to its
           participants.
         *
         * @param participants A HashSet of user-IDs.
         */
        public void updateConversation(HashSet<String>
            participants) {
            boolean exists = false;
            Conversation conversation = null;
            for (Conversation con : user.getConversations()) {
                if (con.getInvolvedUsers().equals(participants))
                     {
                    conversation = con;
                    exists = true;
                }

            }
            if (!exists) {
                conversation = new Conversation(participants);
                addConversation(conversation);
            }
            sendConversation(conversation);
        }

        /**
         * Adds given conversation to all its participants' User
             objects.
         *
         * @param con The conversation to be added.
         */
        public void addConversation(Conversation con) {
            for (User user : registeredUsers) {
                for (String ID : con.getInvolvedUsers()) {
                    if (ID.equals(user.getId())) {
                        user.addConversation(con);
                    }
                }
            }
        }

        /**
         * Check if given message is part of an already existing conversation.
         *
         * @param message The message to be checked.
         * @return Whether given message is part of a conversation or not.
         */
        public Conversation isPartOfConversation(Message message) {
            for (Conversation con : user.getConversations()) {
                if (con.getId() == message.getConversationID()) {
                    return con;
                }
            }
            return null;
        }

        /**
         * Forces connecting users to pick a user that's not already logged in,
         * and updates user database if needed.
         * Announces connected to other connected users.
         */
        public void validateIncomingUser() {
            Object object;
            try {
                object = ois.readObject();
                user = (User) object;
                LOGGER.info("Checking online status for user: " + user.getId());
                while (isUserOnline(user.getId())) {
                    LOGGER.info("User " + user.getId() + " already connected. Asking for new name.");
                    sendObject("Client named " + user.getId()+ " already connected, try again!");
                    // Wait for new user
                    object = ois.readObject();
                    user = (User) object;
                    LOGGER.info("Checking online status for user: " + user.getId());
                }
                if (!isUserInDatabase(user)) {
                    registeredUsers.add(user);
                } else {
                    user = getUser(user.getId());
                }
                oos.writeObject(user);
                server.sendObjectToAll("Client connected: " +
                    user.getId());
                LOGGER.info("Client connected: " + user.getId())
                    ;
                sendConnectedClients();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        /**
         * Listens to incoming Messages, Conversations, HashSets
             of User IDs or server messages.
         */
        public void startCommunication() {
            Object object;
            Message message;
            try {
                while (!Thread.interrupted()) {
                    object = ois.readObject();
                    if (object instanceof Message) {
                        message = (Message) object;
                        server.sendMessage(message);
                    } else if (object instanceof Conversation) {
                        Conversation con = (Conversation) object
                            ;
                        oos.writeObject(con);
                    } else if (object instanceof HashSet) {
                        @SuppressWarnings("unchecked")
                        HashSet<String> participants = (HashSet<
                            String>) object;
                        updateConversation(participants);
                    } else {
                        server.sendObjectToAll(object);
                    }
                }
            } catch (IOException e) {
                disconnectClient();
                e.printStackTrace();
            } catch (ClassNotFoundException e2) {
                e2.printStackTrace();
            }
        }

        public void run() {
            validateIncomingUser();
            startCommunication();
            }
        }
}
```

Listing 1: Server

7.1.2 Startserver.java

```
package chat;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.net.InetAddress;
import java.net.UnknownHostException;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

/**
 * Create an server-panel class.
 */
public class StartServer extends JPanel{
    private JPanel pnlServerCenterFlow = new JPanel(new
        FlowLayout());
    private JPanel pnlServerCenterGrid = new JPanel(new
        GridLayout(1,2,5,5));
    private JPanel pnlServerGrid = new JPanel(new GridLayout
        (2,1,5,5));
    private JPanel pnlServerRunning = new JPanel(new
        BorderLayout());

    private JTextField txtServerPort = new JTextField("3450");
    private JLabel lblServerPort = new JLabel("Port:");
    private JLabel lblServerShowServerIp = new JLabel();
    private JLabel lblWelcome = new JLabel("Create a bIRC server
        ");
    private JLabel lblServerRunning = new JLabel("Server is
        running...");
```

```java
    private JButton btnServerCreateServer = new JButton("Create
        Server");

    private Font fontIpPort = new Font("Sans-Serif",Font.PLAIN
        ,17);
    private Font fontInfo = new Font("Sans-Serif",Font.BOLD|Font
        .ITALIC,20);
    private Font fontWelcome = new Font("Sans-Serif", Font.BOLD
        ,25);
    private Font fontButton = new Font("Sans-Serif", Font.BOLD
        ,18);
    private Server server;

    private BorderLayout br = new BorderLayout();

    public StartServer() {
        lookAndFeel();
        initPanels();
        initLabels();
        setlblServerShowServerIp();
        initListeners();
    }

    /**
     * Initiate Server-Panels.
     */
    public void initPanels() {
        setPreferredSize(new Dimension(350,150));
        setOpaque(true);
        setLayout(br);
        setBackground(Color.WHITE);
        add(pnlServerGrid, BorderLayout.CENTER);
        pnlServerGrid.add(pnlServerCenterGrid);
        add(lblServerShowServerIp, BorderLayout.SOUTH);

        pnlServerCenterFlow.setOpaque(true);
        pnlServerCenterFlow.setBackground(Color.WHITE);
        pnlServerCenterGrid.setOpaque(true);
        pnlServerCenterGrid.setBackground(Color.WHITE);
        pnlServerGrid.setOpaque(true);
        pnlServerGrid.setBackground(Color.WHITE);

        pnlServerCenterGrid.add(lblServerPort);
        pnlServerCenterGrid.add(txtServerPort);
        btnServerCreateServer.setFont(fontButton);
        pnlServerGrid.add(btnServerCreateServer);
        pnlServerRunning.add(lblServerRunning, BorderLayout.
            CENTER);
    }

    /**
     * Initiate Server-Labels.
     */
    public void initLabels() {
        lblServerPort.setHorizontalAlignment(JLabel.CENTER);
        lblWelcome.setHorizontalAlignment(JLabel.CENTER );
        lblServerShowServerIp.setFont(fontInfo);
        lblServerShowServerIp.setForeground(new Color(146,1,1));
        lblServerShowServerIp.setHorizontalAlignment(JLabel.
            CENTER);
        lblServerPort.setFont(fontIpPort);
        lblServerPort.setOpaque(true);
        lblServerPort.setBackground(Color.WHITE);
        lblWelcome.setFont(fontWelcome);
        add(lblWelcome, BorderLayout.NORTH);
        txtServerPort.setFont(fontIpPort);
        lblServerRunning.setFont(fontInfo);
    }

    /**
     * Method that shows the user that the server is running.
     */
    public void setServerRunning() {
        remove(br.getLayoutComponent(BorderLayout.CENTER));
        add(lblServerRunning, BorderLayout.CENTER);
        lblServerRunning.setHorizontalAlignment(JLabel.CENTER);
        validate();
        repaint();
    }

    /**
     * Initiate Listeners.
     */
    public void initListeners() {
        CreateStopServerListener create = new
            CreateStopServerListener();
        EnterListener enter = new EnterListener();
        btnServerCreateServer.addActionListener(create);
        txtServerPort.addKeyListener(enter);
    }

    /**
     * Sets the ip-label to the local ip of your own computer.
     */
    public void setlblServerShowServerIp() {
        try {
            String message = ""+ InetAddress.getLocalHost();
            String realmessage[] = message.split("/");
            lblServerShowServerIp.setText("Server ip is: " +
                realmessage[1]);
        } catch (UnknownHostException e) {
            JOptionPane.showMessageDialog(null, "An error
                occurred.");
        }
    }

    /**
     * Main method for create a server-frame.
     * @param args
     */
    public static void main(String[] args) {
        StartServer server = new StartServer();
        JFrame frame = new JFrame("bIRC Server");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.add(server);
        frame.pack();
        frame.setVisible(true);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
    }

    /**
     * Returns the port from the textfield.
     *
     * @return Port for creating a server.
     */
    public int getPort() {
        return Integer.parseInt(this.txtServerPort.getText());
    }

    /**
     * Set the "Look and Feel".
     */
    public void lookAndFeel() {
         try {
                UIManager.setLookAndFeel(UIManager.
                    getSystemLookAndFeelClassName());
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (UnsupportedLookAndFeelException e) {
                e.printStackTrace();
            }
      }

    /**
     * Listener for create server. Starts a new server with the
         port of the textfield.
     */
    private class CreateStopServerListener implements
        ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (btnServerCreateServer==e.getSource()) {
                server = new Server(getPort());
                setServerRunning();
            }
        }
    }

```

```java
    /**
     * Enter Listener for creating a server.
     */
    private class EnterListener implements KeyListener {
        public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                server = new Server(getPort());
                setServerRunning();
            }
        }

        public void keyReleased(KeyEvent arg0) {}

        public void keyTyped(KeyEvent arg0) {}
    }
}
```

Listing 2: StartServer

7.2 Klient

7.2.1 ChatWindow.java

```java
package chat;

import java.awt.BorderLayout;
import java.awt.Color;

import javax.swing.*;
import javax.swing.text.*;

/**
 * Class used to present content in the main window.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class ChatWindow extends JPanel {
    private int ID;
    private JScrollPane scrollPane;
    private JTextPane textPane;

    private SimpleAttributeSet chatFont = new SimpleAttributeSet
        ();
    private SimpleAttributeSet nameFont = new SimpleAttributeSet
        ();

    /**
     * Constructor that takes an ID from a Conversation, and
       creates a window to display it.
     *
     * @param ID The Conversation object's ID.
     */
    public ChatWindow(int ID) {
        setLayout(new BorderLayout());
        this.ID = ID;
        textPane = new JTextPane();
        scrollPane = new JScrollPane(textPane);

        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
        scrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

        StyleConstants.setForeground(chatFont, Color.BLACK);
        StyleConstants.setFontSize(chatFont, 20);

        StyleConstants.setForeground(nameFont, Color.BLACK);
        StyleConstants.setFontSize(nameFont, 20);
        StyleConstants.setBold(nameFont, true);

        add(scrollPane, BorderLayout.CENTER);
        textPane.setEditable(false);
    }

    /**
     * Appends a new message into the panel window.
     * The message can either contain a String or an ImageIcon.
     *
     * @param message The message object which content will be displayed.
     */
    public void append(final Message message) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                StyledDocument doc = textPane.getStyledDocument();
                try {
                    doc.insertString(doc.getLength(), message.getTimestamp() + " - ", chatFont);
                    doc.insertString(doc.getLength(), message.getFromUserID() + ": ", nameFont);
                    if (message.getContent() instanceof String) {
                        doc.insertString(doc.getLength(), (String)message.getContent(), chatFont);
                    } else {
                        ImageIcon icon = (ImageIcon)message.getContent();
                        StyleContext context = new StyleContext();
                        Style labelStyle = context.getStyle(StyleContext.DEFAULT_STYLE);
                        JLabel label = new JLabel(icon);
                        StyleConstants.setComponent(labelStyle, label);
                        doc.insertString(doc.getLength(), " Ignored", labelStyle);
                    }
                    doc.insertString(doc.getLength(), "\n", chatFont);
                    textPane.setCaretPosition(textPane.getDocument().getLength());

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

    /**
     * Appends a string into the panel window.
     *
     * @param stringMessage The string to be appended.
     */
    public void append(String stringMessage) {
        StyledDocument doc = textPane.getStyledDocument();
        try {
            doc.insertString(doc.getLength(), "[Server: " + stringMessage + "]\n", chatFont);
        } catch (BadLocationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Returns the ChatWindow's ID.
     *
     * @return The ChatWindow's ID.
     */
    public int getID() {
        return ID;
    }
}
```

Listing 3: ChatWindow

7.2.2 Client.java

```java
package chat;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketTimeoutException;
import java.util.ArrayList;

import javax.swing.JOptionPane;

/**
 * Model class for the client.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */

public class Client {
    private Socket socket;
    private ClientController controller;
    private ObjectInputStream ois;
    private ObjectOutputStream oos;
    private User user;
    private String name;
    private static long delay = System.currentTimeMillis();


    /**
     * Constructor that creates a new Client with given ip, port
         and user name.
     *
     * @param ip The IP address to connect to.
     * @param port Port used in the connection.
     * @param name The user name to connect with.
     */
    public Client(String ip, int port, String name) {
        this.name = name;
        try {
            socket = new Socket(ip, port);
            ois = new ObjectInputStream(socket.getInputStream())
                ;
            oos = new ObjectOutputStream(socket.getOutputStream
                ());
            controller = new ClientController(this);
            new ClientListener().start();
        } catch (IOException e) {
            System.err.println(e);
            if (e.getCause() instanceof SocketTimeoutException)
                {

            }
        }
    }

    /**
     * Sends an object object to the server.
     *
     * @param object The object that should be sent to the
         server.
     */
    public void sendObject(Object object) {
        try {
            delay = System.currentTimeMillis() - delay;
            oos.writeObject(object);
            oos.flush();
        } catch (IOException e) {}
    }

    /**
     * Sets the client user by creating a new User object with
       given name.
     *
     * @param name The name of the user to be created.
     */
    public void setName(String name) {
        user = new User(name);
    }

    /**
     * Returns the clients User object.
     *
     * @return The clients User object.
     */
    public User getUser() {
        return user;
    }

    /**
     * Closes the clients socket.
     */
    public void disconnectClient() {
        try {
            socket.close();
        } catch (Exception e) {}
    }

    /**
     * Sends the users conversations to the controller to be
       displayed in the UI.
     */
    public void initConversations() {
        for (Conversation con : user.getConversations()) {
            controller.newConversation(con);
        }
    }

    /**
     * Asks for a username, creates a User object with given
       name and sends it to the server.
     * The server then either accepts or denies the User object.
     * If successful, sets the received User object as current
       user and announces login in chat.
     * If not, notifies in chat and requests a new name.
     */
    public synchronized void setUser() {
        Object object = null;
        setName(this.name);
        while (!(object instanceof User)) {
            try {
                sendObject(user);
                object = ois.readObject();
                if (object instanceof User) {
                    user = (User)object;
                    controller.newMessage("You logged in as " +
                        user.getId());
                    initConversations();

                } else {
                    controller.newMessage(object);
                    this.name = JOptionPane.showInputDialog("
                        Pick a name: ");
                    setName(this.name);
                }
            } catch (IOException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e2) {
                e2.printStackTrace();
            }

        }
    }

    /**
     * Listens to incoming Messages, user lists, Conversations
         or server messages, and deal with them accordingly.
     */
    public void startCommunication() {
        Object object;
        try {
            while (!Thread.interrupted()) {
                object = ois.readObject();
                if (object instanceof Message) {

                    controller.newMessage(object);
                } else if (object instanceof ArrayList) {
                    ArrayList<String> userList = (ArrayList<
                        String>) object;
                    controller.setConnectedUsers(userList);
                } else if (object instanceof Conversation) {
                    Conversation con = (Conversation)object;
                    user.addConversation(con);
                    controller.newConversation(con);
                } else {
                    controller.newMessage(object);
                }
            }
        } catch (IOException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e2) {
                e2.printStackTrace();
            }
        }

        /**
         * Class to handle communication between client and server.
         */
        private class ClientListener extends Thread {
            public void run() {
                setUser();
                startCommunication();
            }
        }
}
```

Listing 4: Client

7.2.3 ClientController.java

```
package chat;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.HashSet;

/**
 * Controller class to handle system logic between client and
    GUI.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class ClientController {
    private ClientUI ui = new ClientUI(this);
    private Client client;

    /**
     * Creates a new Controller (with given Client).
     * Also creates a new UI, and displays it in a JFrame.
     *
     * @param client
     */
    public ClientController(Client client) {
        this.client = client;
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                JFrame frame = new JFrame("bIRC");
                frame.setDefaultCloseOperation(JFrame.
                    EXIT_ON_CLOSE);
                frame.add(ui);
                frame.pack();
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
                ui.focusTextField();
            }
        });
    }

    /**
     * Receives an object that's either a Message object or a
        String
     * and sends it to the UI.
     *
     * @param object A Message object or a String
     */
    public void newMessage(Object object) {
        if (object instanceof Message) {
            Message message = (Message)object;
            ui.appendContent(message);
        } else {
            ui.appendServerMessage((String)object);
        }
    }

    /**
     * Returns the current user's ID.
     *
     * @return A string containing the current user's ID.
     */
    public String getUserID () {
        return client.getUser().getId();
    }

    /**
     * Creates a new message containing given ID and content,
        then sends it to the client.
     *
     * @param conID Conversation-ID of the message.
     * @param content The message's content.
     */
    public void sendMessage(int conID, Object content) {
        Message message = new Message(conID, client.getUser().
            getId(), content);
        client.sendObject(message);
    }

    /**
     * Takes a conversation ID and String with URL to image,
        scales the image and sends it to the client.
     *
     * @param conID Conversation-ID of the image.
     * @param url A string containing the URl to the image to be
         sent.
     */
    public void sendImage(int conID, String url) {
        ImageIcon icon = new ImageIcon(url);
        Image img = icon.getImage();
        BufferedImage scaledImage = ImageScaleHandler.
            createScaledImage(img, 250);
        icon = new ImageIcon(scaledImage);
        sendMessage(conID, icon);
    }


    /**
     * Creates a HashSet of given String array with participants
         , and sends it to the client.
     *
     * @param conversationParticipants A string array with
         conversaion participants.
     */
    public void sendParticipants(String[]
        conversationParticipants) {
        HashSet<String> setParticpants = new HashSet<>();
        for(String participant: conversationParticipants) {
            setParticpants.add(participant);
        }
        client.sendObject(setParticpants);
    }

    /**
     * Sends the ArrayList with connected users to the UI.
     *
     * @param userList The ArrayList with connected users.
     */
    public void setConnectedUsers(ArrayList<String> userList) {
        ui.setConnectedUsers(userList);
    }

    /**
     * Presents a Conversation in the UI.
     *
     * @param con The Conversation object to be presented in the
         UI.
     */
    public void newConversation(Conversation con) {
        HashSet<String> users = con.getInvolvedUsers();
        String[] usersHashToStringArray = users.toArray(new
            String[users.size()]);
        int conID = con.getId();
        ui.createConversation(usersHashToStringArray, conID);
        for (Message message : con.getConversationLog()) {
            ui.appendContent(message);
        }
    }
}
```

Listing 5: ClientController

7.2.4 ClientUI.java

```java
package chat;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.io.File;
import java.util.ArrayList;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.JTextPane;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.text.BadLocationException;
import javax.swing.text.DefaultCaret;
import javax.swing.text.SimpleAttributeSet;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyledDocument;

/**
 * Viewer class to handle the GUI.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */

public class ClientUI extends JPanel {
    private JPanel southPanel = new JPanel();
    private JPanel eastPanel = new JPanel();
    private JPanel eastPanelCenter = new JPanel(new BorderLayout
        ());
    private JPanel eastPanelCenterNorth = new JPanel(new
        FlowLayout());
    private JPanel pnlGroupSend = new JPanel(new GridLayout
        (1,2,8,8));
    private JPanel pnlFileSend = new JPanel(new BorderLayout
        (5,5));
    private long delay = 0;

    private String userString = "";
    private int activeChatWindow = -1;
    private boolean createdGroup = false;

    private JLabel lblUser = new JLabel();
    private JButton btnSend = new JButton("Send");
    private JButton btnNewGroupChat = new JButton();
    private JButton btnLobby = new JButton("Lobby");
    private JButton btnCreateGroup = new JButton("");
    private JButton btnFileChooser = new JButton();

    private JTextPane tpConnectedUsers = new JTextPane();
    private ChatWindow cwLobby = new ChatWindow(-1);
    private ClientController clientController;
    private GroupPanel groupPanel;

    private JTextField tfMessageWindow = new JTextField();
    private BorderLayout bL = new BorderLayout();

    private JScrollPane scrollConnectedUsers = new JScrollPane(
        tpConnectedUsers);
    private JScrollPane scrollChatWindow = new JScrollPane(
        cwLobby);
    private JScrollPane scrollGroupRooms = new JScrollPane(
        eastPanelCenterNorth);

    private JButton[] groupChatList = new JButton[20];
    private ArrayList<JCheckBox> arrayListCheckBox = new
        ArrayList<JCheckBox>();
    private ArrayList<ChatWindow> arrayListChatWindows = new
        ArrayList<ChatWindow>();

    private Font txtFont = new Font("Sans-Serif", Font.BOLD ,
        20);
    private Font fontGroupButton = new Font("Sans-Serif",Font.
        PLAIN, 12);
    private Font fontButtons = new Font("Sans-Serif", Font.BOLD
        ,15);
    private SimpleAttributeSet chatFont = new SimpleAttributeSet
        ();

    public ClientUI(ClientController clientController) {
        this.clientController = clientController;
        arrayListChatWindows.add(cwLobby);
        groupPanel = new GroupPanel();
        groupPanel.start();
```

```java
        lookAndFeel();
        initGraphics();
        initListeners();
    }

    /**
     * Initiates graphics and design.
     * Also initiates the panels and buttons.
     */
    public void initGraphics() {
        setLayout(bL);
        setPreferredSize(new Dimension(900,600));
        eastPanelCenterNorth.setPreferredSize(new Dimension
            (130,260));
        initScroll();
        initButtons();
        add(scrollChatWindow, BorderLayout.CENTER);
        southPanel();
        eastPanel();
    }

    /**
     * Initiates the butons.
     * Also sets the icons and the design of the buttons.
     */
    public void initButtons() {
        btnNewGroupChat.setIcon(new ImageIcon("src/resources/
            newGroup.png"));
        btnNewGroupChat.setBorder(null);
        btnNewGroupChat.setPreferredSize(new Dimension(64,64));

        btnFileChooser.setIcon(new ImageIcon("src/resources/
            newImage.png"));
        btnFileChooser.setBorder(null);
        btnFileChooser.setPreferredSize(new Dimension(64, 64));

        btnLobby.setFont(fontButtons);
        btnLobby.setForeground(new Color(1,48,69));
        btnLobby.setBackground(new Color(201,201,201));
        btnLobby.setOpaque(true);
        btnLobby.setBorderPainted(false);

        btnCreateGroup.setFont(fontButtons);
        btnCreateGroup.setForeground(new Color(1,48,69));
    }

    /**
     * Initiates the scrollpanes and styleconstants.
     */
    public void initScroll() {
        scrollChatWindow.setVerticalScrollBarPolicy(JScrollPane.
            VERTICAL_SCROLLBAR_AS_NEEDED);
        scrollChatWindow.setHorizontalScrollBarPolicy(
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        scrollConnectedUsers.setVerticalScrollBarPolicy(
            JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
        scrollConnectedUsers.setHorizontalScrollBarPolicy(
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        DefaultCaret caretConnected = (DefaultCaret)
            tpConnectedUsers.getCaret();
        caretConnected.setUpdatePolicy(DefaultCaret.
            ALWAYS_UPDATE);
        tpConnectedUsers.setEditable(false);

        tfMessageWindow.setFont(txtFont);
        StyleConstants.setForeground(chatFont, Color.BLACK);
        StyleConstants.setBold(chatFont, true);
    }

    /**
     * Requests that tfMessageWindow gets focus.
     */
    public void focusTextField() {
        tfMessageWindow.requestFocusInWindow();
    }

    /**
     * Initialises listeners.
     */
    public void initListeners() {
        tfMessageWindow.addKeyListener(new EnterListener());
        GroupListener groupListener = new GroupListener();
        SendListener sendListener = new SendListener();
        LobbyListener disconnectListener = new LobbyListener();
        btnNewGroupChat.addActionListener(groupListener);
        btnCreateGroup.addActionListener(groupListener);
        btnLobby.addActionListener(disconnectListener);
        btnFileChooser.addActionListener(new FileChooserListener
            ());
        btnSend.addActionListener(sendListener);
    }

    /**
     * The method takes a ArrayList of the connected users and
         sets the user-checkboxes and
     * the connected user textpane based on the users in the
         ArrayList.
     *
     * @param connectedUsers The ArrayList of the connected
         users.
     */
    public void setConnectedUsers(ArrayList<String>
        connectedUsers) {
        setUserText();
        tpConnectedUsers.setText("");
        updateCheckBoxes(connectedUsers);
        for (String ID : connectedUsers) {
            appendConnectedUsers(ID);
        }
    }

    /**
     * Sets the usertext in the labels to the connected user.
     */
    public void setUserText() {
        lblUser.setText(clientController.getUserID());
        lblUser.setFont(txtFont);
    }

    /**
     * The south panel in the ClientUI BorderLayout.SOUTH.
     */
    public void southPanel() {
        southPanel.setLayout(new BorderLayout());
        southPanel.add(tfMessageWindow, BorderLayout.CENTER);
        southPanel.setPreferredSize(new Dimension(600, 50));

        btnSend.setPreferredSize(new Dimension(134, 40));
        btnSend.setFont(fontButtons);
        btnSend.setForeground(new Color(1, 48, 69));
        southPanel.add(pnlFileSend, BorderLayout.EAST);

        pnlFileSend.add(btnFileChooser, BorderLayout.WEST);
        pnlFileSend.add(btnSend, BorderLayout.CENTER);

        add(southPanel, BorderLayout.SOUTH);
    }

    /**
     * The east panel in ClientUI BorderLayout.EAST.
     */
    public void eastPanel() {
        eastPanel.setLayout(new BorderLayout());
        eastPanel.add(lblUser, BorderLayout.NORTH);
        eastPanel.add(eastPanelCenter, BorderLayout.CENTER);
        eastPanelCenterNorth.add(pnlGroupSend);
        eastPanelCenter.add(scrollGroupRooms, BorderLayout.NORTH
            );
        eastPanelCenter.add(scrollConnectedUsers, BorderLayout.
            CENTER);

        pnlGroupSend.add(btnNewGroupChat);

        eastPanel.add(btnLobby, BorderLayout.SOUTH);
        add(eastPanel, BorderLayout.EAST);
    }

    /**
     * Appends the message to the chatwindow object with the ID
        of the message object.
     *
```

```java
     * @param message The message object with an ID and a
         message.
     */
    public void appendContent(Message message) {

        System.out.println(System.currentTimeMillis() - delay);

        getChatWindow(message.getConversationID()).append(
             message);
        if(activeChatWindow != message.getConversationID()) {
            highlightGroup(message.getConversationID());
        }
    }

    /**
     * The method handles notice.
     *
     * @param ID The ID of the group.
     */
    public void highlightGroup(int ID) {
        if(ID != -1)
            groupChatList[ID].setBackground(Color.PINK);
    }

    /**
     * Appends the string content in the chatwindow-lobby.
     *
     * @param content Is a server message
     */
    public void appendServerMessage(String content) {
        cwLobby.append(content.toString());
    }

    /**
     * The method updates the ArrayList of checkboxes and add
         the checkboxes to the panel.
     * Also checks if the ID is your own ID and doesn't add a
         checkbox of yourself.
     * Updates the UI.
     *
     * @param checkBoxUserIDs ArrayList of UserID's.
     */
    public void updateCheckBoxes(ArrayList<String>
        checkBoxUserIDs) {
        arrayListCheckBox.clear();
        groupPanel.pnlNewGroup.removeAll();
        for (String ID : checkBoxUserIDs) {
            if (!ID.equals(clientController.getUserID())) {
                arrayListCheckBox.add(new JCheckBox(ID));
            }
        }
        for (JCheckBox box: arrayListCheckBox) {
            groupPanel.pnlNewGroup.add(box);
        }
        groupPanel.pnlOuterBorderLayout.revalidate();
    }

    /**
     * The method appends the text in the textpane of the connected users.
     *
     * @param message Is a username.
     */
    public void appendConnectedUsers(String message){
        StyledDocument doc = tpConnectedUsers.getStyledDocument();
        try {
            doc.insertString(doc.getLength(), message + "\n", chatFont);
        } catch (BadLocationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Sets the text on the groupbuttons to the users you check in the checkbox.
     * Adds the new group chat connected with a button and a ChatWindow.
     * Enables you to change rooms.
     * Updates UI.
     *
     * @param participants String-Array of the participants of the new groupchat.
     * @param ID The ID of the participants of the new groupchat.
     */
    public void createConversation(String[] participants, int ID) {
        GroupButtonListener gbListener = new GroupButtonListener();
        for (int i = 0; i < participants.length; i++) {
            if (!(participants[i].equals(clientController.getUserID()))) {
                if (i == participants.length - 1) {
                    userString += participants[i];
                }else {
                    userString += participants[i] + " " ;
                }
            }
        }
        if (ID < groupChatList.length && groupChatList[ID] == null) {
            groupChatList[ID] = (new JButton(userString));
            groupChatList[ID].setPreferredSize(new Dimension(120,30));
            groupChatList[ID].setOpaque(true);
            groupChatList[ID].setBorderPainted(false);
            groupChatList[ID].setFont(fontGroupButton);
            groupChatList[ID].setForeground(new Color(93,0,0));
            groupChatList[ID].addActionListener(gbListener);

            eastPanelCenterNorth.add(groupChatList[ID]);

            if (getChatWindow(ID)==null) {
                arrayListChatWindows.add(new ChatWindow(ID));
            }

            eastPanelCenterNorth.revalidate();
            if (createdGroup) {
                if (activeChatWindow == -1) {
                    btnLobby.setBackground(null);
                }
                else {
                    groupChatList[activeChatWindow].
                        setBackground(null);
                }

                groupChatList[ID].setBackground(new Color
                    (201,201,201));
                remove(bL.getLayoutComponent(BorderLayout.CENTER
                    ));
                add(getChatWindow(ID), BorderLayout.CENTER);
                activeChatWindow = ID;
                validate();
                repaint();
                createdGroup = false;
            }
        }
        this.userString = "";
    }

    /**
     * Sets the "Look and Feel" of the panels.
     */
    public void lookAndFeel() {
        try {
            UIManager.setLookAndFeel(UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (UnsupportedLookAndFeelException e) {
            e.printStackTrace();
        }
    }

    /**
```

```
     * The method goes through the ArrayList of chatwindow
       object and
     * returns the correct one based on the ID.
     *
     * @param ID The ID of the user.
     * @return ChatWindow A ChatWindow object with the correct
       ID.
     */
    public ChatWindow getChatWindow(int ID) {
        for(ChatWindow cw : arrayListChatWindows) {
            if(cw.getID() == ID) {
                return cw;
            }
        }
        return null;
    }

    /**
     * The class extends Thread and handles the Create a group
       panel.
     */
    private class GroupPanel extends Thread {
        private JFrame groupFrame;
        private JPanel pnlOuterBorderLayout = new JPanel(new
            BorderLayout());
        private JPanel pnlNewGroup = new JPanel();
        private JScrollPane scrollCheckConnectedUsers = new
            JScrollPane(pnlNewGroup);

        /**
         * The metod returns the JFrame groupFrame.
         *
         * @return groupFrame
         */
        public JFrame getFrame() {
            return groupFrame;
        }

        /**
         * Runs the frames of the groupPanes.
         */
        public void run() {
            panelBuilder();
            groupFrame = new JFrame();
            groupFrame.setDefaultCloseOperation(JFrame.
                DISPOSE_ON_CLOSE);
            groupFrame.add(pnlOuterBorderLayout);
            groupFrame.pack();
            groupFrame.setVisible(false);
            groupFrame.setLocationRelativeTo(null);
        }

        /**
```

```
         * Initiates the scrollpanels and the panels of the
            groupPanel.
         */
        public void panelBuilder() {
            scrollCheckConnectedUsers.setVerticalScrollBarPolicy
                (JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
            scrollCheckConnectedUsers.
                setHorizontalScrollBarPolicy(JScrollPane.
                HORIZONTAL_SCROLLBAR_NEVER);
            btnCreateGroup.setText("New Conversation");
            pnlOuterBorderLayout.add(btnCreateGroup,
                BorderLayout.SOUTH);
            pnlOuterBorderLayout.add(scrollCheckConnectedUsers,
                BorderLayout.CENTER);
            scrollCheckConnectedUsers.setPreferredSize(new
                Dimension(200,500));
            pnlNewGroup.setLayout(new GridLayout(100,1,5,5));
        }
    }

    /**
     * KeyListener for the messagewindow.
     * Enables you to send a message with enter.
     */
    private class EnterListener implements KeyListener {
        public void keyPressed(KeyEvent e) {
            if (e.getKeyCode() == KeyEvent.VK_ENTER && !(
                tfMessageWindow.getText().isEmpty())) {
                    clientController.sendMessage(
                        activeChatWindow, tfMessageWindow.getText
                        ());
                    tfMessageWindow.setText("");
            }
        }

        public void keyReleased(KeyEvent e) {}

        public void keyTyped(KeyEvent e) {}
    }

    /**
     * Listener that listens to New Group Chat-button and the
         Create Group Chat-button.
     * If create group is pressed, a new button will be created
         with the right name,
     * the right participants.
     * The method use alot of ArrayLists of checkboxes,
         participants and strings.
     * Also some error-handling with empty buttons.
     */
    private class GroupListener implements ActionListener {
        private ArrayList<String> participants = new ArrayList<
            String>();
        private String[] temp;
        public void actionPerformed(ActionEvent e) {
            if (btnNewGroupChat == e.getSource() &&
                arrayListCheckBox.size() > 0) {
                groupPanel.getFrame().setVisible(true);
            }
            if (btnCreateGroup == e.getSource()) {
                participants.clear();
                temp = null;
                for(int i = 0; i < arrayListCheckBox.size(); i
                    ++) {
                    if(arrayListCheckBox.get(i).isSelected()) {
                        participants.add(arrayListCheckBox.get(i
                            ).getText());
                    }
                }

                temp = new String[participants.size() + 1];
                temp[0] = clientController.getUserID();
                for (int i = 1; i <= participants.size(); i++) {
                    temp[i] = participants.get(i-1);
                }
                if (temp.length > 1) {
                    clientController.sendParticipants(temp);
                    groupPanel.getFrame().dispose();
                    createdGroup = true;
                } else {
                    JOptionPane.showMessageDialog(null, "You
                        have to choose atleast one person!");
                }
            }
        }
    }

    /**
     * Listener that connects the right GroupChatButton in an
         ArrayList to the right
     * active chat window.
     * Updates the UI.
     */
    private class GroupButtonListener implements ActionListener
        {
        public void actionPerformed(ActionEvent e) {
            for(int i = 0; i < groupChatList.length; i++) {
                if(groupChatList[i]==e.getSource()) {
                    if(activeChatWindow == -1) {
                        btnLobby.setBackground(null);
                    }
                    else {
                        groupChatList[activeChatWindow].
                            setBackground(null);
                    }
                    groupChatList[i].setBackground(new Color
                        (201,201,201));
                    remove(bL.getLayoutComponent(BorderLayout.
                        CENTER));
                    add(getChatWindow(i), BorderLayout.CENTER);
                    activeChatWindow = i;
                    validate();
                    repaint();
                }
            }
        }
    }

    /**
     * Listener that connects the user with the lobby chatWindow
         through the Lobby button.
     * Updates UI.
     */
    private class LobbyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (btnLobby==e.getSource()) {
                btnLobby.setBackground(new Color(201,201,201));
                if(activeChatWindow != -1)
                    groupChatList[activeChatWindow].
                        setBackground(null);
                remove(bL.getLayoutComponent(BorderLayout.CENTER
                    ));
                add(getChatWindow(-1), BorderLayout.CENTER);
                activeChatWindow = -1;
                invalidate();
                repaint();
            }
        }
    }

    /**
     * Listener that creates a JFileChooser when the button
        btnFileChooser is pressed.
     * The JFileChooser is for images in the chat and it calls
        the method sendImage in the controller.
     */
    private class FileChooserListener implements ActionListener
        {
        public void actionPerformed(ActionEvent e) {
            if (btnFileChooser==e.getSource()) {
                JFileChooser fileChooser = new JFileChooser();
                int returnValue = fileChooser.showOpenDialog(
                    null);
                if (returnValue == JFileChooser.APPROVE_OPTION)
                    {
                    File selectedFile = fileChooser.
                        getSelectedFile();
                    String fullPath = selectedFile.
                        getAbsolutePath();
                    clientController.sendImage(activeChatWindow,
                         fullPath);
                }
            }
        }
    }

    /**
     * Listener for the send message button.
     * Resets the message textfield text.
     */
    private class SendListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (btnSend==e.getSource() && !(tfMessageWindow.
                getText().isEmpty())) {
                    delay = System.currentTimeMillis();
                    clientController.sendMessage(
                        activeChatWindow, tfMessageWindow.getText
                        ());
                    tfMessageWindow.setText("");
            }
        }
    }
}
```

Listing 6: ClientUI

7.2.5 ImageScaleHandler.java

```
package chat;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

import org.imgscalr.Scalr;
import org.imgscalr.Scalr.Method;

/**
 * Scales down images to preferred size.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class ImageScaleHandler {

    private static BufferedImage toBufferedImage(Image img) {
        if (img instanceof BufferedImage) {
            return (BufferedImage) img;
```

```java
        }
        BufferedImage bimage = new BufferedImage(img.getWidth(
            null),
                img.getHeight(null), BufferedImage.TYPE_INT_ARGB
                    );
        Graphics2D bGr = bimage.createGraphics();
        bGr.drawImage(img, 0, 0, null);
        bGr.dispose();
        return bimage;
    }

    public static BufferedImage createScaledImage(Image img, int
         height) {
        BufferedImage bimage = toBufferedImage(img);
        bimage = Scalr.resize(bimage, Method.ULTRA_QUALITY,
                Scalr.Mode.FIT_TO_HEIGHT, 0, height);
        return bimage;
    }

    // Example
    public static void main(String[] args) {
        ImageIcon icon = new ImageIcon("src/filer/new1.jpg");
        Image img = icon.getImage();

        // Use this to scale images
        BufferedImage scaledImage = ImageScaleHandler.
            createScaledImage(img, 75);
        icon = new ImageIcon(scaledImage);

        JLabel lbl = new JLabel();
        lbl.setIcon(icon);
        JPanel panel = new JPanel();
        panel.add(lbl);
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

Listing 7: ImageScaleHandler

7.2.6 StartClient.java

```java
package chat;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;

/**
 * Log in UI and start-class for the chat.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson.
 */
public class StartClient extends JPanel {
    private JLabel lblIp = new JLabel("IP:");
    private JLabel lblPort = new JLabel("Port:");
    private JLabel lblWelcomeText = new JLabel("Log in to bIRC")
        ;
    private JLabel lblUserName = new JLabel("Username:");

    private JTextField txtIp = new JTextField("localhost");
    private JTextField txtPort = new JTextField("3450");
    private JTextField txtUserName = new JTextField();

    private JButton btnLogIn = new JButton("Login");
    private JButton btnCancel = new JButton("Cancel");

    private Font fontWelcome = new Font("Sans-Serif",Font.BOLD
        ,25);
    private Font fontIpPort = new Font("Sans-Serif",Font.PLAIN
        ,17);
    private Font fontButtons = new Font("Sans-Serif",Font.BOLD
        ,15);
    private Font fontUserName = new Font("Sans-Serif",Font.BOLD
        ,17);

    private JPanel pnlCenterGrid = new JPanel(new GridLayout
        (3,2,5,5));
    private JPanel pnlCenterFlow = new JPanel(new FlowLayout());
    private JPanel pnlNorthGrid = new JPanel(new GridLayout
        (2,1,5,5));
    private JPanel pnlNorthGridGrid = new JPanel(new GridLayout
        (1,2,5,5));

    private JFrame frame;

    public StartClient() {
        setLayout(new BorderLayout());
        initPanels();
        lookAndFeel();
        initGraphics();
        initButtons();
        initListeners();
        frame = new JFrame("bIRC Login");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.add(this);
        frame.pack();
        frame.setVisible(true);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
    }

    /**
     * Initiates the listeners.
     */
    public void initListeners() {
        LogInMenuListener log = new LogInMenuListener();
        btnLogIn.addActionListener(log);
        txtUserName.addActionListener(new EnterListener());
        btnCancel.addActionListener(log);
    }

    /**
     * Initiates the panels.
     */
    public void initPanels(){
        setPreferredSize(new Dimension(400, 180));
        pnlCenterGrid.setBounds(100, 200, 200, 50);
        add(pnlCenterFlow,BorderLayout.CENTER);
        pnlCenterFlow.add(pnlCenterGrid);

        add(pnlNorthGrid,BorderLayout.NORTH);
        pnlNorthGrid.add(lblWelcomeText);
        pnlNorthGrid.add(pnlNorthGridGrid);
        pnlNorthGridGrid.add(lblUserName);
        pnlNorthGridGrid.add(txtUserName);

        lblUserName.setHorizontalAlignment(JLabel.CENTER);
        lblUserName.setFont(fontIpPort);
        lblWelcomeText.setHorizontalAlignment(JLabel.CENTER);
        lblWelcomeText.setFont(fontWelcome);
        lblIp.setFont(fontIpPort);
        lblPort.setFont(fontIpPort);
    }

    /**
     * Initiates the buttons.
     */
    public void initButtons() {
        btnCancel.setFont(fontButtons);
        btnLogIn.setFont(fontButtons);

        pnlCenterGrid.add(lblIp);
        pnlCenterGrid.add(txtIp);
        pnlCenterGrid.add(lblPort);
        pnlCenterGrid.add(txtPort);
        pnlCenterGrid.add(btnLogIn);
        pnlCenterGrid.add(btnCancel);
    }

    /**
     * Initiates the graphics and some design.
     */
    public void initGraphics() {
        pnlCenterGrid.setOpaque(false);
        pnlCenterFlow.setOpaque(false);
        pnlNorthGridGrid.setOpaque(false);
        pnlNorthGrid.setOpaque(false);
        setBackground(Color.WHITE);
        lblUserName.setBackground(Color.WHITE);
        lblUserName.setOpaque(false);

        btnLogIn.setForeground(new Color(41,1,129));
        btnCancel.setForeground(new Color(41,1,129));

        txtIp.setFont(fontIpPort);
        txtPort.setFont(fontIpPort);
        txtUserName.setFont(fontUserName);
    }

    /**
     * Sets the "Look and Feel" of the JComponents.
     */
    public void lookAndFeel() {
     try {
            UIManager.setLookAndFeel(UIManager.
                getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (UnsupportedLookAndFeelException e) {
            e.printStackTrace();
        }
   }

    /**
     * Main method for the login-frame.
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                StartClient ui = new StartClient();
            }
        });

    }

    /**
```

```
     * Listener for login-button, create server-button and for the cancel-button.
     * Also limits the username to a 10 char max.
     */
    private class LogInMenuListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (btnLogIn==e.getSource()) {
                    if (txtUserName.getText().length() <= 10) {
                        new Client(txtIp.getText(), Integer.parseInt(txtPort.getText()), txtUserName.getText());
                    } else {
                    JOptionPane.showMessageDialog(null, "Namnet får max vara 10 karaktärer!");
                    txtUserName.setText("");
                }
            }
            if (btnCancel==e.getSource()) {
                System.exit(0);
            }
        }
    }

    /**
     * Listener for the textField. Enables you to press enter instead of login.
     * Also limits the username to 10 chars.
     */
    private class EnterListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if(txtUserName.getText().length() <= 10) {
                new Client(txtIp.getText(), Integer.parseInt(txtPort.getText()),txtUserName.getText());
                frame.dispose();
            } else {
                JOptionPane.showMessageDialog(null, "Namnet får max vara 10 karaktärer!");
                txtUserName.setText("");
            }
        }
    }
}
```

Listing 8: LoginUI

7.3 Delade klasser

7.3.1 ChatLog

```
package chat;
import java.io.Serializable;
import java.util.Iterator;
import java.util.LinkedList;
```

```java

/**
 * Class to hold logged messages.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */

public class ChatLog implements Iterable<Message>, Serializable
    {
    private LinkedList<Message> list = new LinkedList<Message>()
        ;
    private static int MESSAGE_LIMIT = 30;
    private static final long serialVersionUID =
        13371337133732526L;


    /**
     * Adds a new message to the chat log.
     *
     * @param message The message to be added.
     */
    public void add(Message message) {
        if(list.size() >= MESSAGE_LIMIT) {
            list.removeLast();
        }
        list.add(message);
    }

    public Iterator<Message> iterator(){
        return list.iterator();
    }
}
```

Listing 9: ChatLog

7.3.2 Message

```java
package chat;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;

/**
 * Model class to handle messages
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class Message implements Serializable {
    private String fromUserID;
    private Object content;
    private String timestamp;
    private int conversationID = -1;    /* -1 means it's a lobby
         message */
    private static final long serialVersionUID = 133713371337L;

    /**
     * Constructor that creates a new message with given
         conversation ID, String with information who sent it,
         and its content.
     *
     * @param conversationID The conversation ID.
     * @param fromUserID A string with information who sent the
         message.
     * @param content The message's content.
     */
    public Message(int conversationID, String fromUserID, Object
         content) {
        this.conversationID = conversationID;
        this.fromUserID = fromUserID;
        this.content = content;
        newTime();
    }

    /**
     * Creates a new timestamp for the message.
     */
    private void newTime() {
        Date time = new Date();
        SimpleDateFormat ft = new SimpleDateFormat("HH:mm:ss");
        this.timestamp = ft.format(time);
    }

    /**
     * Returns a string containing sender ID.
     *
     * @return A string with the sender ID.
     */
    public String getFromUserID() {
        return fromUserID;
    }

    /**
     * Returns an int with the conversation ID.
     *
     * @return An int with the conversation ID.
     */
    public int getConversationID() {
        return conversationID;
    }

    /**
     * Returns the message's timestamp.
     *
```

```java
     * @return The message's timestamp.
     */
    public String getTimestamp() {
        return this.timestamp;
    }

    /**
     * Returns the message's content.
     *
     * @return The message's content.
     */
    public Object getContent() {
        return content;
    }
}
```

Listing 10: Message

7.3.3 User

```java
package chat;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Class to hold information of a user.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class User implements Serializable {
    private static final long serialVersionUID = 1273274782824L;
    private ArrayList<Conversation> conversations;
    private String id;

    /**
     * Constructor to create a User with given ID.
     *
     * @param id A string with the user ID.
     */
    public User(String id) {
        this.id = id;
        conversations = new ArrayList<>();
    }

    /**
     * Returns an ArrayList with the user's conversations
     *
     * @return The user's conversations.
     */
    public ArrayList<Conversation> getConversations() {
        return conversations;
    }

    /**
     * Adds a new conversation to the user.
     *
     * @param conversation The conversation to be added.
     */
    public void addConversation(Conversation conversation) {
        conversations.add(conversation);
    }

    /**
     * Returns the user's ID.
     *
     * @return The user's ID.
     */
    public String getId() {
        return id;
    }
}
```

Listing 11: User

7.3.4 Conversation

```java
package chat;

import java.io.Serializable;
import java.util.HashSet;

/**
 * Class to hold information of a conversation.
 *
 * @author Emil Sandgren, Kalle Bornemark, Erik Sandgren,
 * Jimmy Maksymiw, Lorenz Puskas & Rasmus Andersson
 */
public class Conversation implements Serializable {
    private HashSet<String> involvedUsers;
    private ChatLog conversationLog;
    private int id;
    private static int numberOfConversations = 0;

    /**
     * Constructor that takes a HashSet of involved users.
     *
     * @param involvedUsersID The user ID's to be added to the
           conversation.
     */
    public Conversation(HashSet<String> involvedUsersID) {
        this.involvedUsers = involvedUsersID;
        this.conversationLog = new ChatLog();
        id = ++numberOfConversations;
    }

    /**
     * Returns a HashSet of the conversation's involved users.
     *
     * @return A hashSet of the conversation's involved users.
     */
    public HashSet<String> getInvolvedUsers() {
        return involvedUsers;
    }

    /**
     * Returns the conversion's ChatLog.
     *
     * @return The conversation's ChatLog.
     */
    public ChatLog getConversationLog() {
        return conversationLog;
    }

    /**
     * Adds a message to the conversation.
     *
     * @param message The message to be added.
     */
    public void addMessage(Message message) {
        conversationLog.add(message);

    }

    /**
     * Return the conversation's ID.
     *
     * @return The conversation's ID.
     */
    public int getId() {
        return id;
    }

}
```

Listing 12: Conversation