Gruppuppgift DA343A VT19
Grupp 7

Medlemmar
Anders Bergh
Amanda Eriksson
Andreas Jönsson
Haris Obradovac
Patrik Skuza
Filip Spånberg

Innehållsförteckning

Arbetsbeskrivning

Anders Bergh
Anders skrev koden för klasserna SerializableImage, PacketLogger samt LogViewer tillsammans med Haris. Anders har även ansvar för sekvensdiagrammet som visar vad som sker när logg mellan två tidpunkter visas i gränssnittet.

Amanda Eriksson
Amanda har skrivit klasserna ClientConnection samt ServerConnection. Amanda ansvarar för alla klassdiagram samt sekvensdiagrammet som visar vad som sker när en klient skickar ett meddelande till servern.

Andreas Jönsson
Andreas ansvarar för sekvensdiagramet uppdatering av anslutna användare. Andreas skrev koden för ServerConnection och Controller.

Haris Obradovac
Haris har skrivit klasserna SerializableImage, PacketLogger och LogViewer tillsammans med Anders. Ansvarar för sekvensdiagrammet som visar när klienter ansluter till systemet.

Patrik Skuza
Patrik ansvarar för sekvensdiagrammet: “2. Vad som sker på klientsidan när en användare startar klienten. Detta fall ska inkludera inläsning av Kontakter.” Patrik skrev koden för klasserna ChatHelper, Message, User och ansvarar även för större delen av Controller.

Filip Spånberg
Har arbetat fram ClientGUI och systemet med ChatPanes för att kunna visa flera olika gruppsamtal, och tillhörande GUI-hjälpklasser. Ansvarar för sekvensdiagrammet vid klientnedkoppling.

Instruktioner för programstart

Katalogen “data” måste finnas i projektroten. I fall det blir problem att starta programmet kan .dat-filerna som finns där behövas raderas.

1. Importera projektet i antingen Eclipse eller IntelliJ IDEA.
2. Starta servern, klass: `javachat.server.ServerControl`.
3. Starta klient, klass: `javachat.client.Controller`.

Systembeskrivning

Vårt meddelandesystem består i grunden av den abstrakta klassen Message, som sedan tre klasser ärver från: ConnectMessage, DisconnectMessage och UserMessage.

När en användare kopplar upp sig skickar den ett ConnectMessage (CM). Det fångas upp av servern som skapar en Client på serversidan (om inte användaren finns sedan innan), och sedan skickar den vidare samma CM till alla andra på servern med en “tunn” lista av användare (endast namn). Den användare som kopplat upp sig får ett CM med en “tjock” lista av användare (med bilder på användare) för att kunna se alla som är uppkopplade.

Ett DisconnectMessage (DM) fungerar på liknande sätt. När en användare vill koppla ner sig så skickar den ett DM, servern sätter användaren som offline och skickar vidare det till de andra användarna. När användaren som skickade iväg DM får tillbaka det är anslutningen avslutad.

Och ett UserMessage är ett meddelande från användaren. Det kan ha mottagare null, vilket då betyder “till alla som är online”, eller specifika mottagare satta. Om mottagare är null kommer servern inte att spara dessa meddelanden till användare som inte är online, utan kommer endast spara de som har specificerat mottagare.

Klientsidans interface bygger på ChatPanes, vilka är flikar i huvudfönstret med olika samtalsgrupper. Main kan alla som är online läsa, och den kan inte stängas. När en skickar meddelande till separata användare (markera en användare i listan, eller fler genom att hålla in Control-knappen) så öppnas en ny flik för varje unik kombination av användare. Men samma grupp användare kan inte ha två olika flikar.

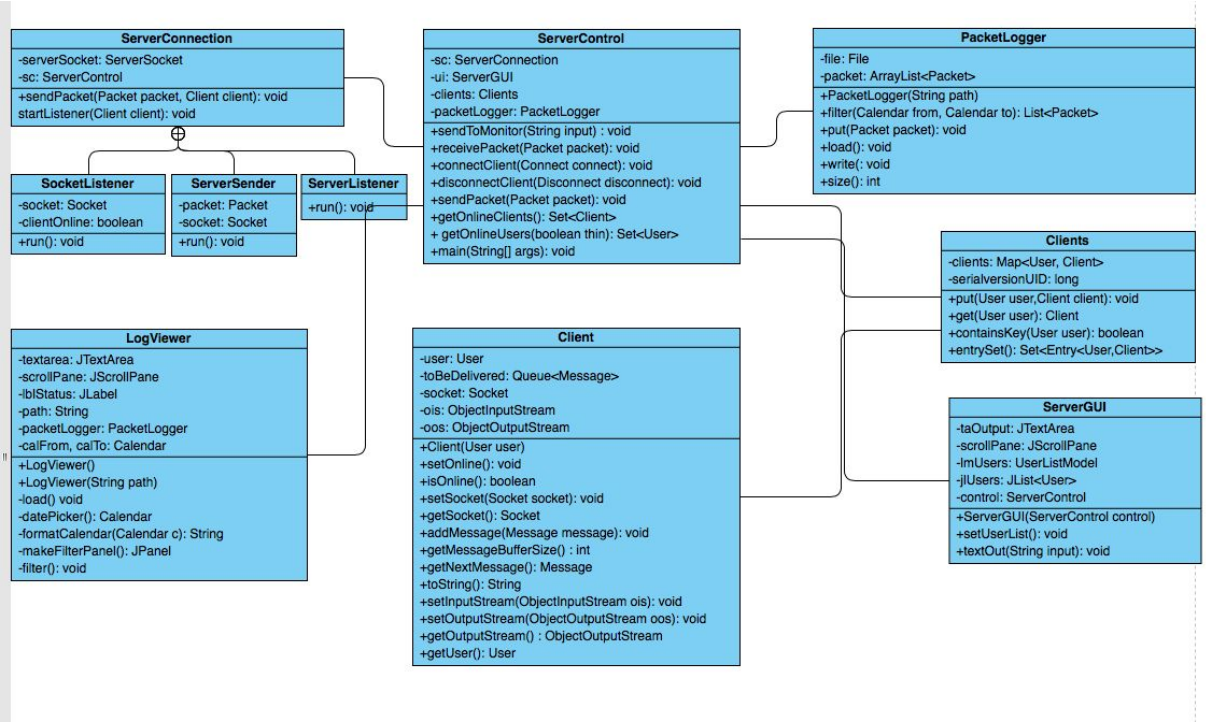
Klientens uppkopplingssida har tre inre klasser, som ärver från Thread. StartConnection startar upp en Output-ström, och startar upp en ClientListener som lyssnar efter meddelanden från servern. Slutligen så startas en ClientSender varje gång ett meddelande ska skickas.

Serversidan startar en tråd direkt som lyssnar efter nya uppkopplingar via en ServerSocket. När den stöter på en uppkoppling så skapar den en ObjectInputStream och tar emot det första meddelandet, som ska vara ett CM. I ServerControl skapas en ny Client utefter avsändaren till meddelandet, OIS läggs till, och en SocketListener-tråd skapas för att lyssna till ytterligare meddelanden från användaren. När meddelanden ska skickas så skapar server liksom klienten en dedikerar tråd för detta.

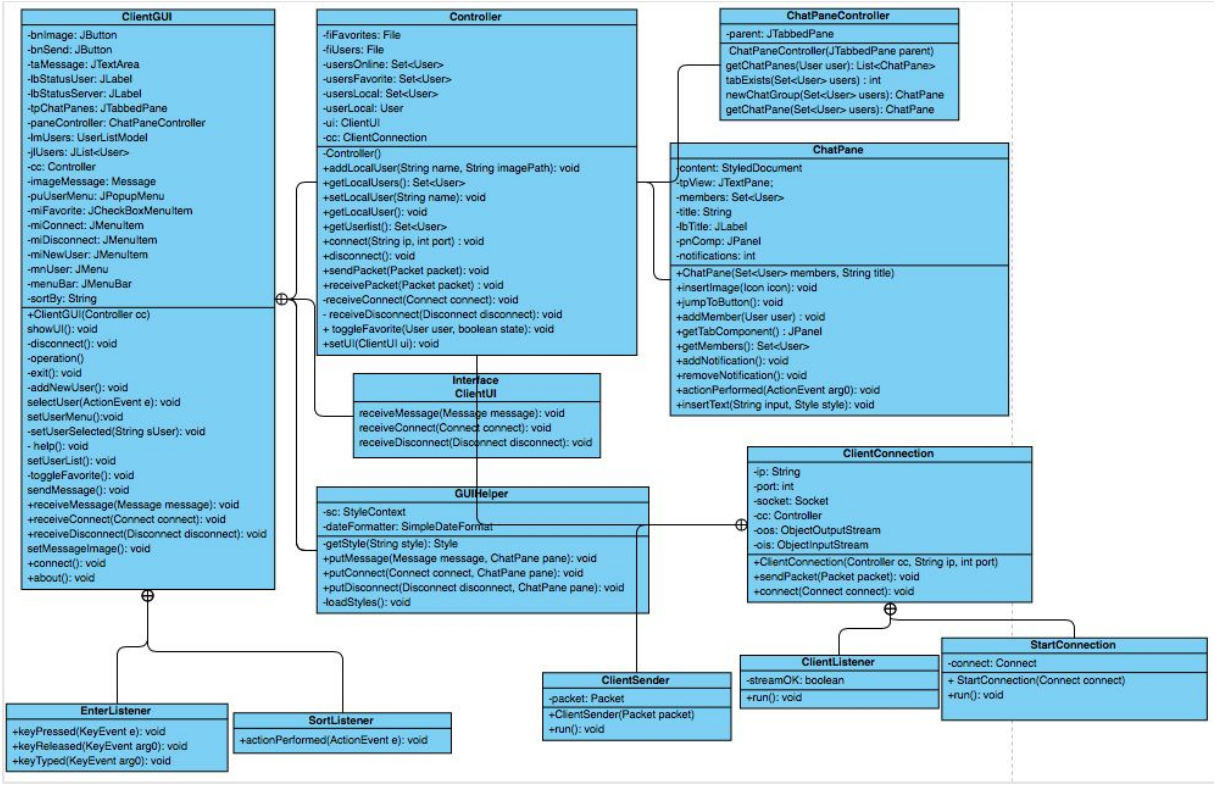
Bilder skickas över objektströmmar med hjälp av klassen SerializableImage istället för ImageIcon, då ImageIcon inte är kompatibelt mellan olika versioner av Java, t.ex. vid kommunikation mellan Java 8 och Java 11.

Klassdiagram

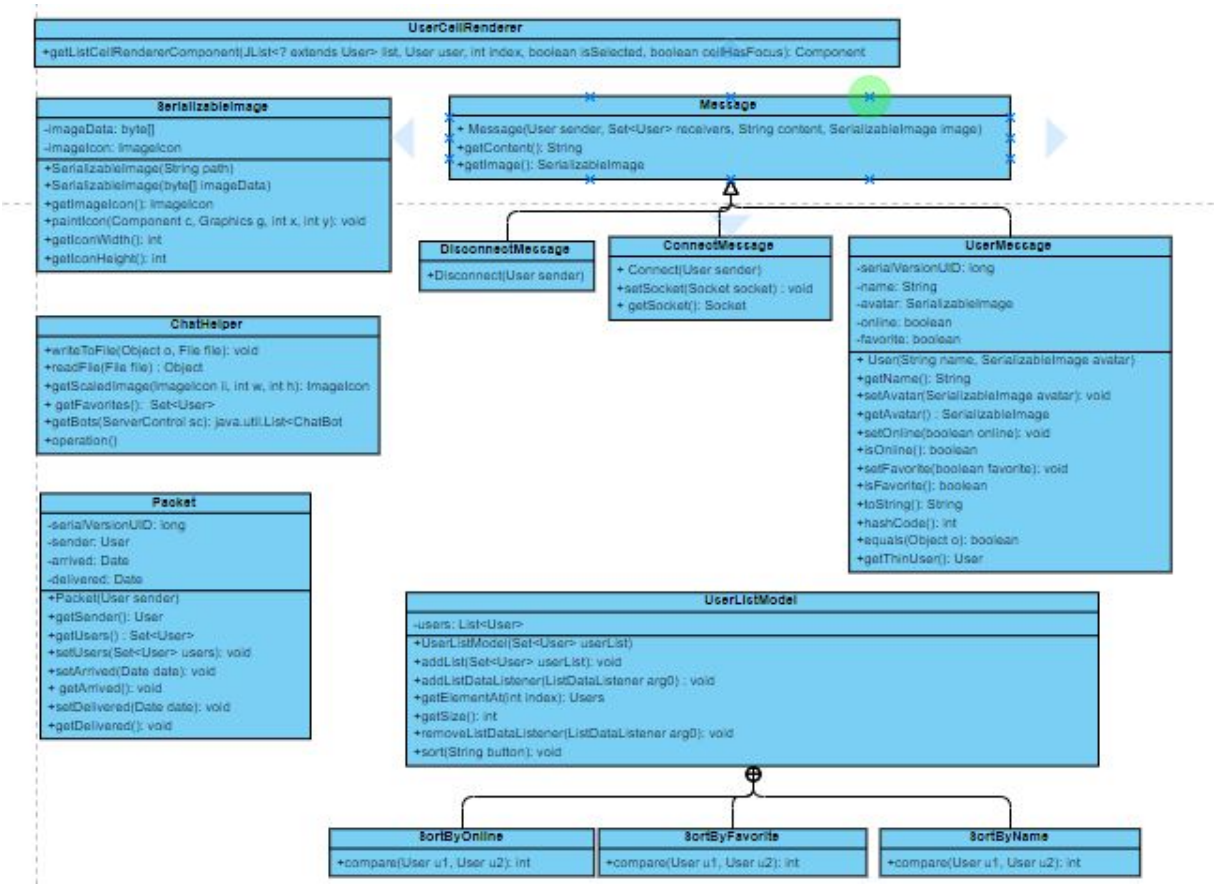
Klassdiagram server



Klassdiagram klient



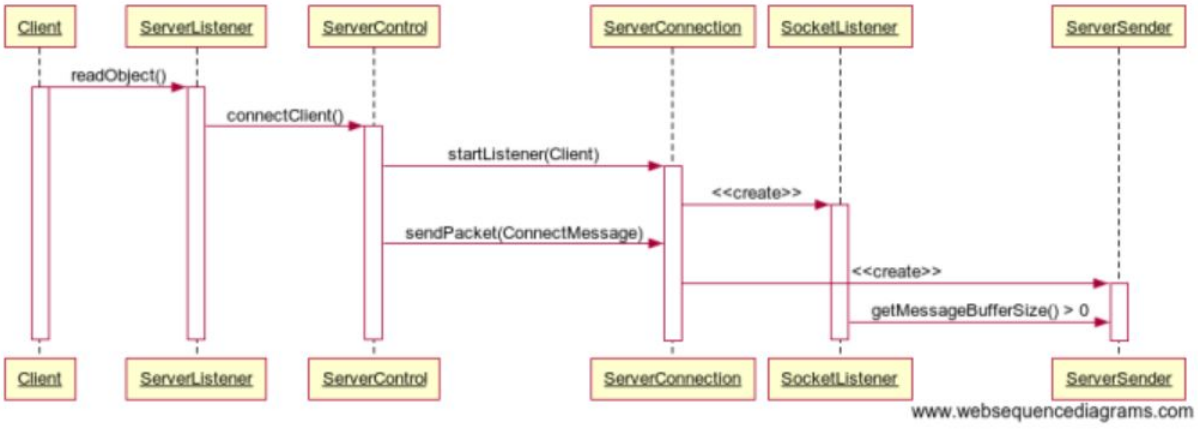
Klassdiagram hjälpklasser



Sekvensdiagram

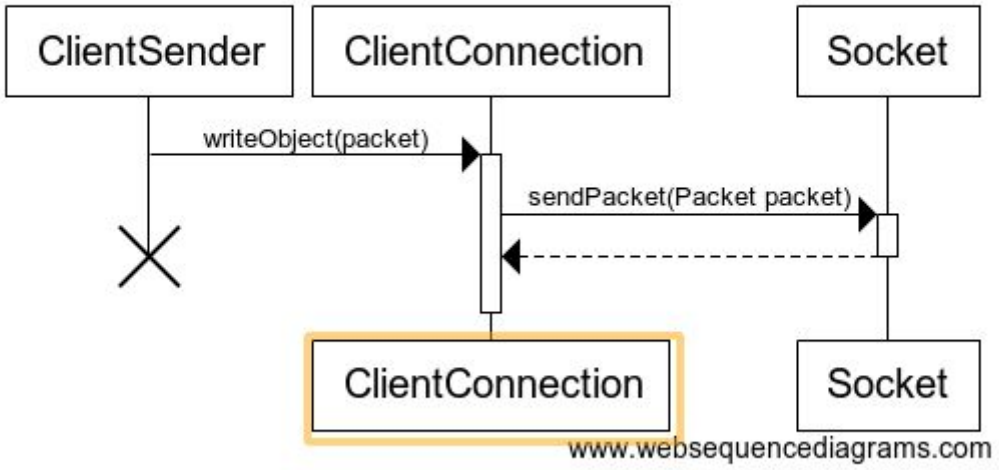
Klient ansluter till systemet

Ansvarig: Haris Obradovac



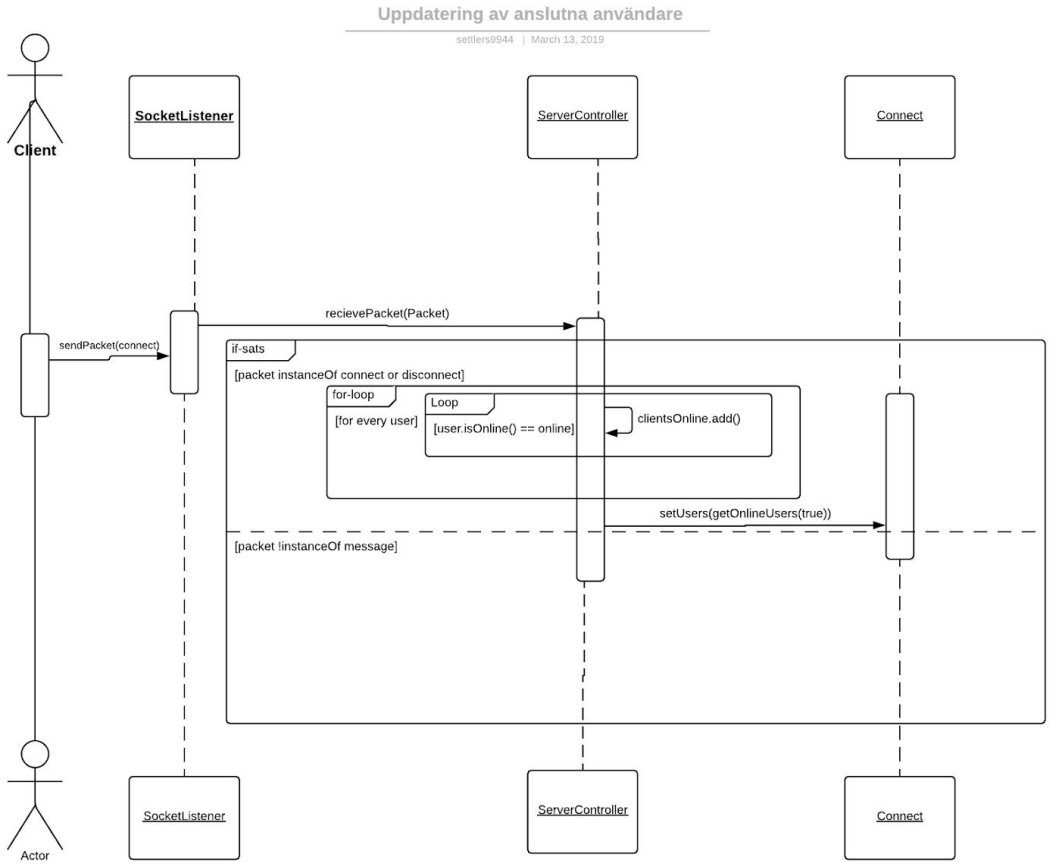
Vad som sker när klient skickar meddelande till server

Ansvarig: Amanda Eriksson



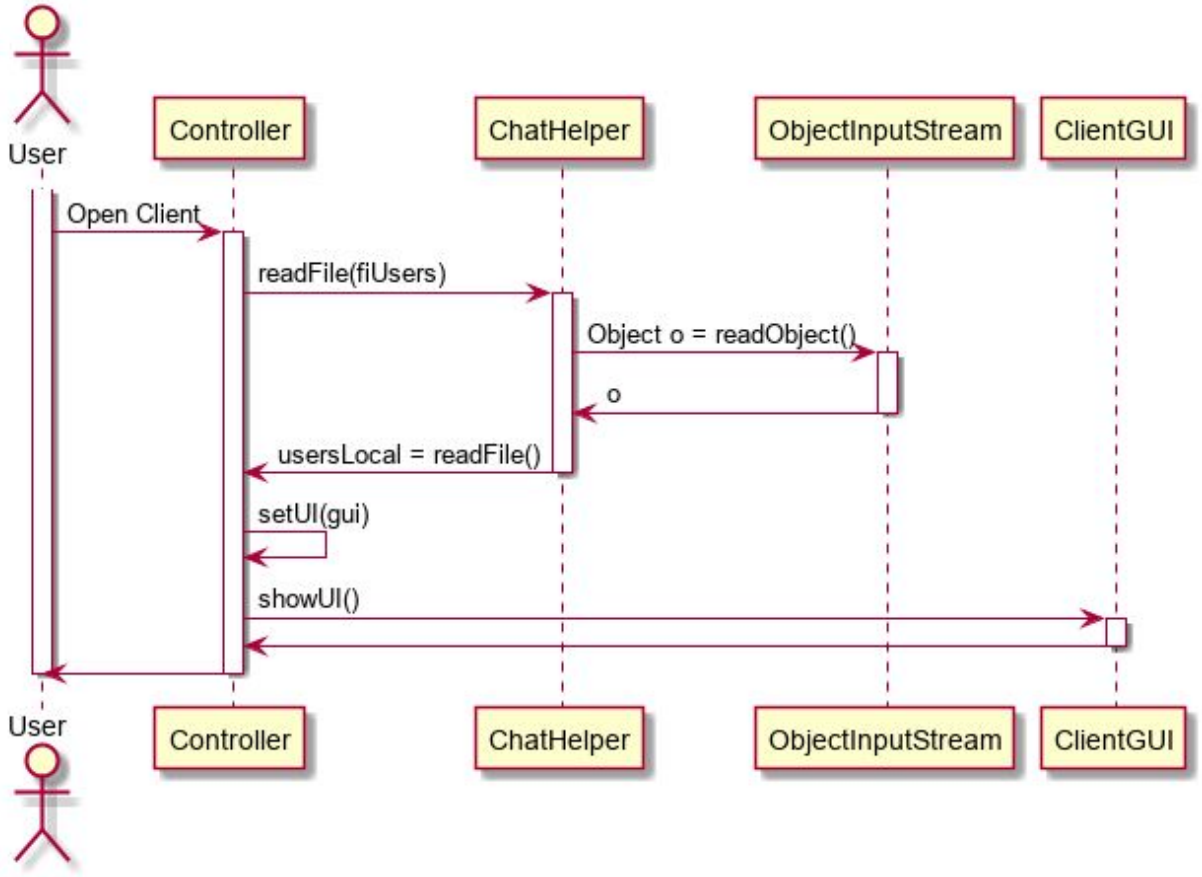
Uppdatering av anslutna användare

Ansvarig: Andreas Jönsson



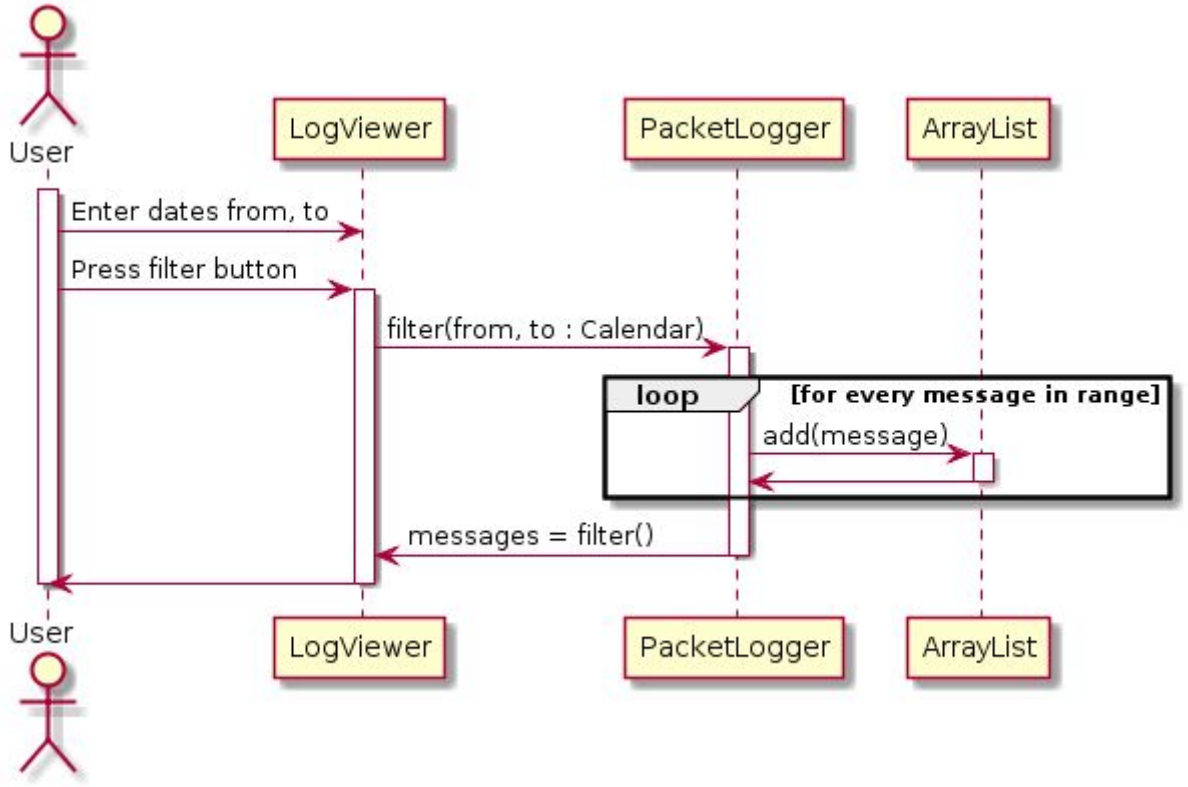
Vad som sker på klientsidan när en användare startar klienten

Ansvarig: Patrik Skuza



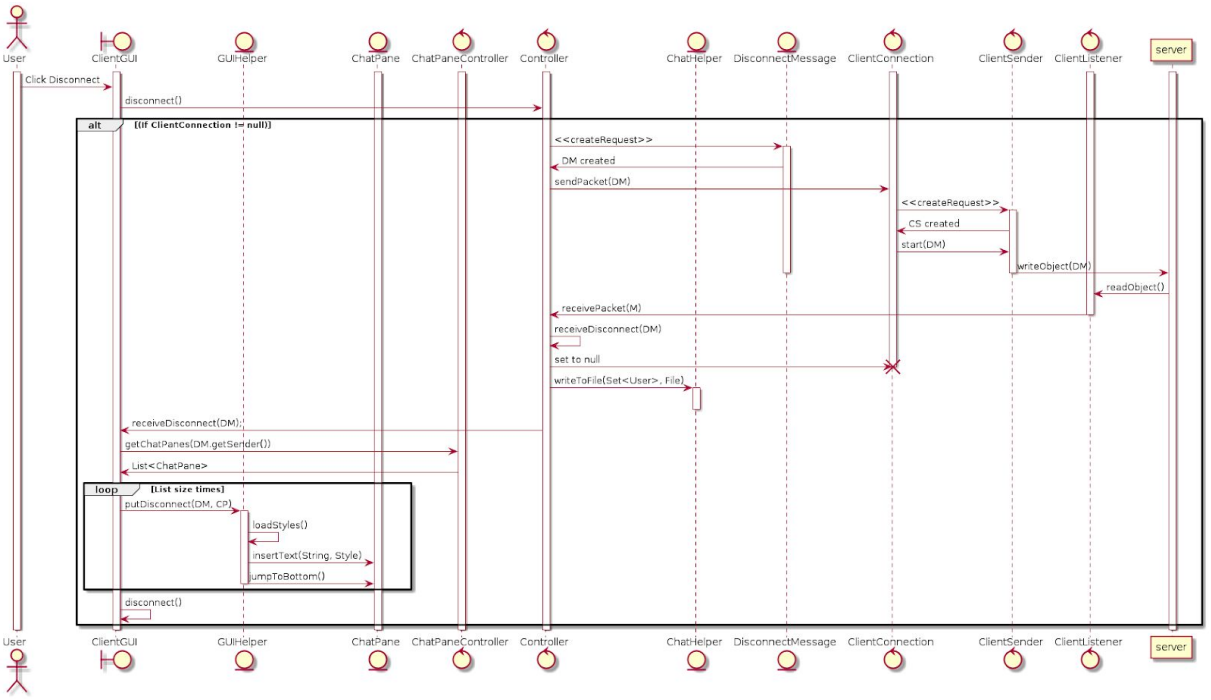
Loggning mellan två tidpunkter i servern

Ansvarig: Anders Bergh



Klient avslutar uppkoppling

Ansvarig: Filip Spånberg



Källkod server

Client

```
package javachat.server;
import javachat.UserMessage;
import javachat.User;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.LinkedList;
import java.util.Queue;

public class Client implements Serializable {
  private User user;
  private Queue<UserMessage> toBeDelivered = new LinkedList<UserMessage>();
  private ObjectInputStream ois;
  private ObjectOutputStream oos;

  private static final long serialversionUID = 2L;

  public Client(User user) {
     this.user = user;
  }

  public User getUser() {
     return user;
  }

  public void setOnline(boolean online) {
     user.setOnline(online);
  }

  public boolean isOnline() {
     return user.isOnline();
  }

  public void setInputStream(ObjectInputStream ois) { this.ois = ois; }

  public ObjectInputStream getInputStream() {
     return ois;
  }

  public void setOutputStream(ObjectOutputStream oos) {
     this.oos = oos;
  }

  public ObjectOutputStream getOutputStream() {
     return oos;
  }

  public synchronized void addMessage(UserMessage userMessage) {
     toBeDelivered.add(userMessage);
  }

  public int getMessageBufferSize() {
     return toBeDelivered.size();
  }

  public synchronized UserMessage getNextMessage() { return toBeDelivered.remove();}

  public String toString() {
     return user.toString();
  }
}
```

Clients

```
package javachat.server;

import javachat.User;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class Clients implements Serializable {
  private static final long serialVersionUID = 1L;
  private Map<User, Client> clients = new HashMap<>();

  public synchronized void put(User user,Client client) {
     clients.put(user,client);
  }

  public synchronized Client get(User user) {
     return clients.get(user);
  }

  public synchronized boolean containsKey(User user) {
     return clients.containsKey(user);
  }

  public synchronized Set<Entry<User,Client>> entrySet() {
     return clients.entrySet();
  }
}
```

LogViewer

```java
package javachat.server;

import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.Message;
import javachat.UserMessage;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.List;

/**
* LogViewer is a program that can be used to inspect packets sent/received.
*
* @author Anders Bergh
*/
public class LogViewer extends JPanel {
   JTextArea textArea = new JTextArea(24, 80);
   JScrollPane scrollPane = new JScrollPane(textArea);
   JLabel lblStatus = new JLabel("Status");
   private String path;
   private PacketLogger packetLogger;
   private Calendar calFrom, calTo;

   /**
    * Creates a LogViewer with a default log path set.
    */
   public LogViewer() {
       this("data/log.dat");
   }

   /**
    * Creates a LogViewer that reads from a specified file.
    *
    * @param path The path to the log file to read.
    */
   public LogViewer(String path) {
       this.path = path;

       setLayout(new BorderLayout(5, 5));
       setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

       textArea.setEditable(false);
       textArea.setFont(new Font("Monospaced", Font.PLAIN, 12));

       add(makeFilterPanel(), BorderLayout.NORTH);
       add(scrollPane, BorderLayout.CENTER);
       add(lblStatus, BorderLayout.SOUTH);

       load();
   }

   public static void main(String[] args) {
       SwingUtilities.invokeLater(() -> {
           JFrame frame = new JFrame("LogViewer");
           frame.add(new LogViewer());
           frame.pack();
           frame.setLocationRelativeTo(null);
           frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           frame.setVisible(true);
       });
   }

   /**
    * Reads the server log and updates the UI to show how many packets were read.
    */
   private void load() {
       packetLogger = new PacketLogger(path);
       lblStatus.setText("Packets loaded: " + packetLogger.size());
   }

   /**
    * Asks for date/time input from the user.
    *
    * @return An instance of Calendar with time set by the user, or null if invalid format/no entry.
    */
   private Calendar datePicker() {
```

```
    Calendar calendar = Calendar.getInstance();
    String input = JOptionPane.showInputDialog("Enter date, format: YYYY-mm-dd HH:mm");

    if (input == null) {
        return null;
    }

    // "YYYY-mm-dd HH:mm".length() = 16
    if (input.length() != 16) {
        JOptionPane.showMessageDialog(null, "Invalid format, expected YYYY-mm-dd HH:mm.");
        return null;
    }

    String sYear = input.substring(0, 4);
    String sMonth = input.substring(5, 7);
    String sDay = input.substring(8, 10);
    String sHour = input.substring(11, 13);
    String sMinute = input.substring(14, 16);

    try {
        calendar.set(Calendar.YEAR, Integer.parseInt(sYear));
        calendar.set(Calendar.MONTH, Integer.parseInt(sMonth) - 1);
        calendar.set(Calendar.DAY_OF_MONTH, Integer.parseInt(sDay));
        calendar.set(Calendar.HOUR_OF_DAY, Integer.parseInt(sHour));
        calendar.set(Calendar.MINUTE, Integer.parseInt(sMinute));
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Invalid input, got:" + e.toString());
        return null;
    }

    return calendar;
}

/**
 * Formats a Calendar object into the format YYYY-mm-dd HH:mm:ss
 *
 * @param c Instance of Calendar
 * @return Formatted date.
 */
private String formatCalendar(Calendar c) {
    return String.format("%04d-%02d-%02d %02d:%02d",
            c.get(Calendar.YEAR),
            c.get(Calendar.MONTH) + 1,
            c.get(Calendar.DAY_OF_MONTH),
            c.get(Calendar.HOUR_OF_DAY),
            c.get(Calendar.MINUTE));
}

private JPanel makeFilterPanel() {
    final JButton
            btnFromDate = new JButton("From"),
            btnToDate = new JButton("To");
    final JTextField
            tfFrom = new JTextField(11),
            tfTo = new JTextField(11);

    ActionListener buttonListener = new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            Calendar c = datePicker();
            if (c == null) {
                return;
            }

            Object src = e.getSource();
            if (src == btnFromDate) {
                tfFrom.setText(formatCalendar(c));
                calFrom = c;
            } else if (src == btnToDate) {
                tfTo.setText(formatCalendar(c));
                calTo = c;
            }
        }
    };

    JPanel panel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 5));

    panel.add(new JLabel("Date range:"));
    tfFrom.setEditable(false);
    panel.add(tfFrom);

    btnFromDate.addActionListener(buttonListener);
    panel.add(btnFromDate);
```

```
        panel.add(new JLabel("-"));

        tfTo.setEditable(false);
        panel.add(tfTo);

        btnToDate.addActionListener(buttonListener);
        panel.add(btnToDate);

        JButton btnFilter = new JButton("Filter");
        btnFilter.addActionListener(e -> filter());
        panel.add(btnFilter);

        JButton btnReloadLog = new JButton("Reload log");
        btnReloadLog.addActionListener(e -> load());
        panel.add(btnReloadLog);

        return panel;
    }

    /**
     * Requests packets received during the specified range, then writes them to the text window.
     */
    private void filter() {
        if (calFrom == null || calTo == null) {
            JOptionPane.showMessageDialog(null, "Set the date range before filtering.");
            return;
        }
        List<Message> messages = packetLogger.filter(calFrom, calTo);
        StringBuilder sb = new StringBuilder();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        for (Message p : messages) {
            sb.append("[");
            sb.append(sdf.format(p.getArrived()));
            sb.append("] ");

            if (p instanceof ConnectMessage) {
                sb.append("ConnectMessage: ");
                sb.append(p.getSender().getName());
                sb.append('\n');
            } else if (p instanceof DisconnectMessage) {
                sb.append("DisconnectMessage: ");
                sb.append(p.getSender().getName());
                sb.append('\n');
            } else if (p instanceof UserMessage) {
                sb.append('<');
                sb.append(p.getSender().getName());
                sb.append("> -> ");
                sb.append(p.getUsers() != null ? p.getUsers().toString() : "All");
                sb.append(": ");
                sb.append(((UserMessage) p).getContent());
                sb.append('\n');
            } else {
                sb.append("Message type: ");
                sb.append(p.getClass().getSimpleName());
                sb.append(", toString() = ");
                sb.append(p.toString());
                sb.append('\n');
            }
        }
        textArea.setText(sb.toString());
    }
}
```

PacketLogger

```java
package javachat.server;

import javachat.Message;

import java.io.*;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

/**
* PacketLogger saves all received messages on disk.
*
* @author Anders Bergh
*/
public class PacketLogger {
   private File file;
   private ArrayList<Message> messages = new ArrayList<>();

   /**
    * Creates a new logger given a path. Reads the log if it exists.
    *
    * @param path Path to the packet log.
    */
   public PacketLogger(String path) {
       file = new File(path);

       load();
   }

   /**
    * Writes a {@link Message} to the log.
    *
    * @param message The message to log.
    */
   public synchronized void put(Message message) {
       messages.add(message);

       // Kanske inte så effektivt att skriva hela loggen varje gång ett paket läggs till, men det är en lösning...
       write();
   }

   /**
    * Retrieves a list of messages matching a date range.
    *
    * @param from Beginning of range.
    * @param to   End of range.
    * @return List of messages.
    */
   public List<Message> filter(Calendar from, Calendar to) {
       ArrayList<Message> list = new ArrayList<>();
       for (Message p : messages) {
           Calendar arrived = Calendar.getInstance();
           arrived.setTime(p.getArrived());
           if ((arrived.equals(from) || arrived.after(from)) && (arrived.equals(to) || arrived.before(to))) {
               list.add(p);
           }
       }
       return list;
   }

   /**
    * Loads the packet log from disk.
    */
   public synchronized void load() {
       if (!file.exists()) {
           return;
       }

       try (FileInputStream fis = new FileInputStream(file);
            BufferedInputStream bis = new BufferedInputStream(fis);
            ObjectInputStream ois = new ObjectInputStream(bis)) {
           ArrayList<Message> obj = (ArrayList<Message>) ois.readObject();
           messages = obj;
       } catch (IOException | ClassNotFoundException e) {
           System.err.println("Unable to load packet log:");
           e.printStackTrace();
       }
   }

   /**
    * Serializes the packet log.
```

```
     */
    public synchronized void write() {
        if (!file.exists()) {
            try {
                file.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        try (FileOutputStream fos = new FileOutputStream(file);
             BufferedOutputStream bos = new BufferedOutputStream(fos);
             ObjectOutputStream oos = new ObjectOutputStream(bos)) {
            oos.writeObject(messages);
        } catch (IOException e) {
            System.err.println("Unable to save packet log:");
            e.printStackTrace();
        }
    }

    /**
     * @return Number of messages logged.
     */
    public int size() {
        return messages.size();
    }
}
```

ServerConnection

```
package javachat.server;

import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.Message;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

// Class for handling connections to users
public class ServerConnection {
   private ServerSocket serverSocket;
   private ServerControl sc;

   public ServerConnection(ServerControl sc, int port) {
       this.sc = sc;
       try {
           serverSocket = new ServerSocket(port);
           sc.sendToMonitor("ServerConnection: Listening on port :" + port);
       } catch (IOException e) {
           sc.sendToMonitor(e.toString());
       }
       Thread sl = new ServerListener();
       sl.start();
   }

   public void sendPacket(Message message, Client client) {
       new ServerSender(message, client).start();
   }

   void startListener(Client client) {
       new SocketListener(client).start();
   }

   // Listens for new connections and sets up ObjectInputStream
   private class ServerListener extends Thread {
       public void run() {
           while (!Thread.interrupted()) {
               try {
                   Socket socket = serverSocket.accept();
                   sc.sendToMonitor(
                           String.format("ServerListener: Heard socket @ IP: %s, port: %d",
                                   socket.getInetAddress().getHostAddress(),
                                   socket.getPort()));
                   ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(socket.getInputStream()));
                   Object o = ois.readObject();
                   if (o instanceof ConnectMessage) {
                       ConnectMessage connect = (ConnectMessage) o;
                       connect.setSocket(socket);
                       sc.connectClient(connect, ois);
                   }
               } catch (Exception e) {
                   sc.sendToMonitor("ServerListener: [" + e + "]");
               }
           }
       }
   }

   // Send Messages to users
   private class ServerSender extends Thread {
       private Message message;
       private Client client;

       public ServerSender(Message message, Client client) {
           this.message = message;
           this.client = client;
       }

       public void run() {
                   sc.sendToMonitor("ServerSender:  sending  "  +  message.getClass().getSimpleName()  +  "  to  "  +
client.getUser());
           try {
               message.setDelivered(new Date());

               synchronized (client.getOutputStream()) {
                   client.getOutputStream().writeObject(message);
                   client.getOutputStream().flush();
```

```
                while (client.getMessageBufferSize() > 0) {
                        sc.sendToMonitor(client.getUser() + " has " + client.getMessageBufferSize() + " messages on
server. Sending...");
                    client.getOutputStream().writeObject(client.getNextMessage());
                    client.getOutputStream().flush();
                }
            }
        } catch (Exception e) {
            sc.sendToMonitor("ServerSender: [" + e.toString() + "]");
        }
    }
}

private class SocketListener extends Thread {
    private Client client;

    public SocketListener(Client client) {
        this.client = client;
    }

    public void run() {
        sc.sendToMonitor("SocketListener: Going to work...");
        while (client.isOnline()) {
            try {
                Message message = (Message) client.getInputStream().readObject();
                sc.sendToMonitor(String.format("SocketListener: Received %s from %s to %s",
                        message.getClass().getSimpleName(),
                        message.getSender(),
                        (message.getUsers() == null ? "all" : message.getUsers())));
                message.setArrived(new Date());
                sc.receivePacket(message);
                if (message instanceof DisconnectMessage)
                    client.setOnline(false);
            } catch (ClassNotFoundException | IOException e) {
                sc.sendToMonitor("SocketListener: [" + e.toString() + "], resetting connection...");
                client.setOnline(false);
            }
        }
        sc.sendToMonitor("SocketListener: Ending. Daisy, Daisy...");
    }
}
```

ServerControl

```
package javachat.server;

import javachat.*;

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

import javax.swing.*;

public class ServerControl {
   private ServerConnection sc;
   private ServerGUI ui = new ServerGUI(this);
   private Clients clients = new Clients();
   private PacketLogger packetLogger = new PacketLogger("data/log.dat");
   private File fiClients = new File("data/clients.dat");

   public ServerControl() {
       sc = new ServerConnection(this, 5561);

       // Reads file of previously connected users
       if (fiClients.exists()) {
           Set<User> users = (Set<User>) ChatHelper.readFile(fiClients);
           if (users == null)
               clients = new Clients();
           else
               for (User u: users) {
                   u.setOnline(false);
                   clients.put(u, new Client(u));
               }
       }

       ui.setUserList();

       SwingUtilities.invokeLater(() -> {
           JFrame frame = new JFrame("Server GUI");
           frame.add(ui);
           frame.pack();
           frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           frame.setVisible(true);
       });
   }

   // Send to server log window
   public void sendToMonitor(String input) {
       ui.textOut(input);
   }

   public void receivePacket(Message message) {
       packetLogger.put(message);
       if (message instanceof UserMessage)
           sendPacket(message);
       else if (message instanceof DisconnectMessage)
           disconnectClient((DisconnectMessage) message);
   }

   // Connects a client to the server
   public synchronized void connectClient(ConnectMessage connect, ObjectInputStream ois) {
       User u = connect.getSender();
       if (!clients.containsKey(u)) { // Check if client already exists
           clients.put(u, new Client(u));
           ChatHelper.writeToFile(getUserlist(), fiClients);
       }
       Client c = clients.get(u);
       c.setOnline(true);
       c.setInputStream(ois);
       try {
                                                                   c.setOutputStream(new      ObjectOutputStream(new
BufferedOutputStream(connect.getSocket().getOutputStream())));
           sendToMonitor("Client's OOS connected: " + c.getOutputStream());
       } catch (IOException e) {
           sendToMonitor("ServerControl: Error creating client's OOS: [" + e + "]");
       }
       sc.startListener(c); // Start listening to Messages from client
       connect.setSocket(null);
       sendToMonitor("User \"" + u + "\" connected.");
       ui.setUserList();

       connect.setUsers(getOnlineUsers(true));
       sendPacket(connect);
```

```
    }

    // Disconnect a client
    public synchronized void disconnectClient(DisconnectMessage disconnect) {
        sendToMonitor("User \"" + disconnect.getSender() + "\" disconnected");
        disconnect.setUsers(getOnlineUsers(true));
        sendPacket(disconnect);
    }

    private void sendPacket(Message message) {
        if (message.getUsers() == null || message instanceof DisconnectMessage) {
            for (Client c : getOnlineClients()) // Send Message to all online clients
                sc.sendPacket(message, c);
            if (message instanceof DisconnectMessage) { // If the message is a Disconnect, set client offline
                clients.get(message.getSender()).setOnline(false);
                ui.setUserList();
            }
        } else if (message instanceof UserMessage) {
            for (User u : message.getUsers()) {
                Client c = clients.get(u);
                if (c.isOnline()) {
                    sendToMonitor("ServerControl: Client online, sending message to ServerConnections");
                    sc.sendPacket(message, c);
                } else {
                    sendToMonitor("ServerControl: Client not online, adding message to buffer...");
                    c.addMessage((UserMessage) message);
                }
            }
        } else if (message instanceof ConnectMessage) { // If Message is user wanting to connect
            ConnectMessage fullConnect = new ConnectMessage(message.getSender()); // Get a full list of online users
            fullConnect.setUsers(getOnlineUsers(false));
            for (Client c : getOnlineClients())
                if (!c.equals(clients.get(message.getSender())))
                    sc.sendPacket(message, c);
                else // and send full list to recently connected user
                    sc.sendPacket(fullConnect, clients.get(message.getSender()));
        }
    }

    private Set<Client> getOnlineClients() {
        Set<Client> clientsOnline = new HashSet<>();
        for (Map.Entry<User, Client> entry : clients.entrySet())
            if (entry.getKey().isOnline())
                clientsOnline.add(entry.getValue());
        return clientsOnline;
    }

    // Get all users
    Set<User> getUserlist() {
        Set<User> users = new HashSet<>();
        for (Map.Entry<User, Client> entry : clients.entrySet())
            users.add(entry.getKey());
        return users;
    }

    // Get online users
    private Set<User> getOnlineUsers(boolean thin) {
        Set<User> users = new HashSet<>();
        for (Map.Entry<User, Client> entry : clients.entrySet())
            if (entry.getKey().isOnline())
                if(thin)
                    users.add(entry.getKey().getThinUser());
                else
                    users.add(entry.getKey());
        return users;
    }

    public static void main(String[] args) {
        ServerControl sControl = new ServerControl();
    }
}
```

ServerGUI

```
package javachat.server;

import javachat.User;
import javachat.UserCellRenderer;
import javachat.UserListModel;

import javax.swing.*;
import javax.swing.text.DefaultCaret;
import java.awt.*;

// Class for monitoring server activity
public class ServerGUI extends JPanel {
   private JTextArea taOutput = new JTextArea(10, 40);
   private JScrollPane scrollPane = new JScrollPane(taOutput);
   private UserListModel lmUsers;
   private JList<User> jlUsers = new JList<>();
   private ServerControl control;

   public ServerGUI(ServerControl control) {
       this.control = control;

       setLayout(new BorderLayout());
       setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
       // Setup user list
       jlUsers.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
       jlUsers.setLayoutOrientation(JList.VERTICAL);
       jlUsers.setCellRenderer(new UserCellRenderer());
       jlUsers.setFixedCellWidth(200);
       JScrollPane spUsers = new JScrollPane(jlUsers);
       spUsers.setPreferredSize(new Dimension(200, 400));
       spUsers.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
       spUsers.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
       add(spUsers, BorderLayout.NORTH);

       // Setup activity monitor
       taOutput.setEditable(false);
       taOutput.setLineWrap(true);
       ((DefaultCaret) taOutput.getCaret()).setUpdatePolicy(DefaultCaret.ALWAYS_UPDATE);

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

       add(scrollPane, BorderLayout.CENTER);
       // Setup traffic search button
       JButton bnTraffic = new JButton("Get traffic");
       bnTraffic.addActionListener(e -> getTraffic());
       add(bnTraffic, BorderLayout.SOUTH);
   }

   // Starts a frame for checking past traffic
   private void getTraffic() {
       JFrame frame = new JFrame("Logger");
       frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
       frame.add(new LogViewer());
       frame.pack();
       frame.setVisible(true);
   }

   synchronized void setUserList() { // Update the user list (for example, when a new user connects)
       lmUsers = new UserListModel(control.getUserlist());
       jlUsers.setModel(lmUsers);
   }

   public void textOut(String input) {
       taOutput.append(input + "\n");
   }
}
```

Källkod klient

ChatPane

```
package javachat.client;

import javachat.User;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Set;

import javax.swing.*;
import javax.swing.text.BadLocationException;
import javax.swing.text.Style;
import javax.swing.text.StyledDocument;

// Class that creates a Chatpane, which is a panel a group of users are members of
public class ChatPane extends JScrollPane implements ActionListener {
  private StyledDocument content;
  private JTextPane tpView;
  private Set<User> members = null;
  private String title = "Main";
  private JLabel lbTitle = new JLabel(title);
  private JPanel pnComp = new JPanel();
  private int notifications = 0;

  // Constructor for main group, where all online users are members
  public ChatPane() {
     super(new JTextPane());
     tpView = (JTextPane)this.getViewport().getView();
     tpView.setLayout(new BorderLayout());
     content = tpView.getStyledDocument();
     tpView.setEditable(false);
     lbTitle.setOpaque(true);
     pnComp.add(lbTitle);
     setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
     setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
  }

  // Constructor for all other panes, with specific members
  public ChatPane(Set<User> members, String title) {
     this();
     this.members = members;
     this.title = title;
     JButton bnClose = new JButton("x");
     bnClose.setMaximumSize(new Dimension(20, 20));
     bnClose.addActionListener(this);
     pnComp.add(bnClose);
  }

  public void insertText(String input, Style style) {
     try {
        content.insertString(content.getLength(), input, style);
     } catch (BadLocationException e) {
        System.out.println(e.toString());
     }
  }

  public void insertImage(Icon icon) {
     tpView.insertIcon(icon);
  }

  // Method to keep the incoming text being seen at the bottom by jumping down the pane
  public void jumpToBottom() {
                                                  if           (getVerticalScrollBar().getValue()           >
getVerticalScrollBar().getMaximum()-(getVerticalScrollBar().getVisibleAmount()+100))
        tpView.setCaretPosition(tpView.getDocument().getLength());
  }

  // Gets the panel at the top of the tab
  public JPanel getTabComponent() {
     return pnComp;
  }

  public Set<User> getMembers() {
     return members;
  }

  public void addNotification() {
     lbTitle.setBackground(new Color(255, 111, 111));
     notifications++;
```

```
      lbTitle.setText(title + "(" + notifications + ")");
   }

   public void removeNotification() {
      lbTitle.setBackground(new Color(238, 238, 238));
      lbTitle.setText(title);
      notifications = 0;
   }

   @Override
   public void actionPerformed(ActionEvent arg0) {
      getParent().remove(this);
   }
}
```

ChatPaneController

```
package javachat.client;

import javachat.User;

import javax.swing.*;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

// Control class for all Chatpanes, to get a pane with a specific
// set of members, och get all panes with a single user
public class ChatPaneController {
   JTabbedPane parent;

   ChatPaneController(JTabbedPane parent) {
       this.parent = parent;
   }

   ChatPane getChatPane(Set<User> users) {
       ChatPane pane;
       if (users == null) {
           pane = (ChatPane) parent.getComponentAt(0);
       } else {
           int tab = tabExists(users);
           if (tab == -1) {
               pane = newChatGroup(users);
           } else {
               pane = (ChatPane) parent.getComponentAt(tab);
           }
       }
       return pane;
   }

   // Get panes user is member of
   List<ChatPane> getChatPanes(User user) {
       List<ChatPane> panes = new LinkedList<>();
       panes.add((ChatPane) parent.getComponentAt(0)); // Add main pain (where members = null)
       int count = parent.getTabCount();
       for (int i = 1; i < count; i++) { // Cycle through the other panes,
           ChatPane pane = ((ChatPane) parent.getComponentAt(i));
           if (pane.getMembers().contains(user))
               panes.add(pane); // And add if user is member, add pane
       }
       return panes;
   }

   // Check if there exists a tab for the group users, and return its tab number if so, else return -1
   int tabExists(Set<User> users) {
       for (int t = 1; t < parent.getTabCount(); t++)
           if (((ChatPane) parent.getComponentAt(t)).getMembers().equals(users))
               return t;
       return -1;
   }

   ChatPane newChatGroup(Set<User> users) {
       StringBuilder title = new StringBuilder();
       for (User u : users)
           title.append(u).append(", ");
       title.delete(title.length() - 2, title.length());
       ChatPane pane = new ChatPane(users, title.toString());
       parent.addTab(null, pane);
       int tabNumber = parent.getTabCount() - 1;
       parent.setTabComponentAt(tabNumber, pane.getTabComponent());
       return pane;
   }
}
```

ClientConnection

```
package javachat.client;

import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.Message;

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

// Class that handles all connections to the server
public class ClientConnection {
   private String ip;
   private int port;
   private Socket socket;
   private Controller cc;
   private ObjectOutputStream oos;
   private ObjectInputStream ois;

   public ClientConnection(Controller cc, String ip, int port) {
       this.ip = ip;
       this.port = port;
       this.cc = cc;
   }

   public void sendPacket(Message message) {
       new ClientSender(message).start();
   }

   public void connect(ConnectMessage connect) {
       new StartConnection(connect).start();
   }

   // Initiates a connection with the server
   private class StartConnection extends Thread {
       private ConnectMessage connect;

       public StartConnection(ConnectMessage connect) {
           this.connect = connect;
       }

       public void run() {
           try {
               System.out.println("Connecting socket...");
               socket = new Socket(ip, port);
               new ClientListener().start();
               System.out.println("Trying to connect output stream...");
               oos = new ObjectOutputStream(new BufferedOutputStream(socket.getOutputStream()));
               System.out.println("Sending connect...");
               sendPacket(connect);
           } catch (IOException e) {
               System.out.println("Error connecting socket: " + e.toString());
           }
       }
   }

   // Listens for Messages from the server
   private class ClientListener extends Thread {
       private boolean streamOK = true;
       public void run() {
           try {
               System.out.println("Trying to connect input stream...");
               ois = new ObjectInputStream(new BufferedInputStream(socket.getInputStream()));
               System.out.println("Input stream connected.");
           } catch (IOException e) {
               System.out.println("ClientListener: Error creating OIS: [" + e + "]");
               streamOK = false;
           }
           while (streamOK) {
               try {
                   Message message = (Message) ois.readObject();
                   message.setDelivered(new Date());
                   System.out.println("Received message " + message.getClass().getSimpleName());
                   cc.receivePacket(message);
               } catch (Exception e) {
                   System.out.println("Connection error: [" + e.toString() + "]");
                   streamOK = false;
                   DisconnectMessage dc = new DisconnectMessage(cc.getLocalUser());
                   dc.setDelivered(new Date());
                   cc.receivePacket(dc);
               }
```

```
            }
        }
    }

    // Sends Messages to the server
    private class ClientSender extends Thread {
        private Message message;

        public ClientSender(Message message) {
            this.message = message;
        }

        public void run() {
            try {
                oos.writeObject(message);
                System.out.println("Message sent.");
                oos.flush();
            } catch (Exception e) {
                System.out.println("ClientSender: " + e.toString());
            }
        }
    }
}
```

ClientGUI

```
package javachat.client;

import javachat.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.border.BevelBorder;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;

class ClientGUI extends JPanel implements ClientUI {
   private JButton bnImage = new JButton("Select image");
   private JButton bnSend = new JButton("Send");
   private JTextArea taMessage = new JTextArea();
   private JLabel lbStatusUser = new JLabel("User not set");
   private JLabel lbStatusServer = new JLabel();
   private JTabbedPane tpChatPanes = new JTabbedPane();
   private ChatPaneController paneController = new ChatPaneController(tpChatPanes);
   private UserListModel lmUsers;
   private JList<User> jlUsers = new JList<>();
   private Controller cc;
   private String imageMessage = null;
   private JPopupMenu puUserMenu = new JPopupMenu();
   private JCheckBoxMenuItem miFavorite = new JCheckBoxMenuItem("Favorite");
   private JMenuItem miConnect = new JMenuItem("Connect");
   private JMenuItem miDisconnect = new JMenuItem("Disconnect");
   private JMenuItem miNewUser = new JMenuItem("New user");
   private JMenu mnUser = new JMenu("User");
   private JMenuBar menuBar = new JMenuBar();
   private User userInList;
   private String sortBy = "Name";

   ClientGUI(Controller cc) {
       this.cc = cc;
       setLayout(new BorderLayout());
       setPreferredSize(new Dimension(600, 400));

       // Setup input area
       JPanel pnInput = new JPanel();
       pnInput.setLayout(new BorderLayout());
       JPanel pnButtons = new JPanel();
       pnButtons.setLayout(new GridLayout(1, 2));
       bnImage.addActionListener(e -> setMessageImage());
       bnImage.setEnabled(false);
       pnButtons.add(bnImage);
       bnSend.addActionListener(e -> sendMessage());
       bnSend.setEnabled(false);
       pnButtons.add(bnSend);
       taMessage.setPreferredSize(new Dimension(500, 80));
       taMessage.setBorder(BorderFactory.createLineBorder(Color.black));
       taMessage.addKeyListener(new EnterListener());
       pnInput.add(taMessage, BorderLayout.CENTER);
       pnInput.add(pnButtons, BorderLayout.EAST);
       add(pnInput, BorderLayout.SOUTH);

       // Setup main chatroom
       ChatPane cpMain = new ChatPane();
       tpChatPanes.addTab(null, cpMain);
       tpChatPanes.setTabComponentAt(0, cpMain.getTabComponent());
       tpChatPanes.addChangeListener(e -> ((ChatPane) tpChatPanes.getSelectedComponent()).removeNotification());
       add(tpChatPanes, BorderLayout.CENTER);

       // Setup user list
       jlUsers.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
       jlUsers.setLayoutOrientation(JList.VERTICAL);
       jlUsers.setCellRenderer(new UserCellRenderer());
       jlUsers.setFixedCellWidth(220);
       JScrollPane spUsers = new JScrollPane(jlUsers);
       spUsers.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
       spUsers.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
```

```
        add(spUsers, BorderLayout.EAST);

        // Setup user list context menu
        JMenuItem miSort = new JMenuItem("Sort by:");
        miSort.setEnabled(false);
        miFavorite.addActionListener(e -> toggleFavorite());
        puUserMenu.add(miFavorite);
        puUserMenu.add(miSort);
        ButtonGroup bnSortGroup = new ButtonGroup();
        JRadioButtonMenuItem miSortName = new JRadioButtonMenuItem("Name");
        JRadioButtonMenuItem miSortOnli = new JRadioButtonMenuItem("Online");
        JRadioButtonMenuItem miSortFavo = new JRadioButtonMenuItem("Favorite");
        bnSortGroup.add(miSortName);
        bnSortGroup.add(miSortFavo);
        bnSortGroup.add(miSortOnli);
        puUserMenu.add(miSortName);
        puUserMenu.add(miSortFavo);
        puUserMenu.add(miSortOnli);
        SortListener sortListener = new SortListener();
        miSortName.addActionListener(sortListener);
        miSortFavo.addActionListener(sortListener);
        miSortOnli.addActionListener(sortListener);
        miSortName.setSelected(true);
        jlUsers.addMouseListener(new MouseAdapter() {
            public void mouseReleased(MouseEvent e) {
                if (SwingUtilities.isRightMouseButton(e)) {
                    int index = jlUsers.locationToIndex(e.getPoint());
                    userInList = lmUsers.getElementAt(index);
                    miFavorite.setState(userInList.isFavorite());
                    puUserMenu.show(e.getComponent(), e.getX(), e.getY());
                }
            }
        });

        // Setup menu
        JMenu mnFile = new JMenu("File");
        mnFile.setMnemonic(KeyEvent.VK_F);
        menuBar.add(mnFile);

        miConnect.setMnemonic(KeyEvent.VK_C);
        miConnect.addActionListener(e -> connect());
        miDisconnect.setEnabled(false);
        miDisconnect.setMnemonic(KeyEvent.VK_D);
        miDisconnect.addActionListener(e -> disconnect());
        miConnect.setEnabled(false);
        JMenuItem miExit = new JMenuItem("Exit");
        miExit.addActionListener(e -> exit());
        mnFile.add(miConnect);
        mnFile.add(miDisconnect);
        mnFile.addSeparator();
        mnFile.add(miExit);

        mnUser.setMnemonic(KeyEvent.VK_U);
        menuBar.add(mnUser);
        miNewUser.addActionListener(e -> addNewUser());
        mnUser.add(miNewUser);

        JMenu mnHelp = new JMenu("Help");
        mnHelp.setMnemonic(KeyEvent.VK_H);
        menuBar.add(mnHelp);

        JMenuItem miHelp = new JMenuItem("Help");
        mnHelp.setMnemonic(KeyEvent.VK_H);
        miHelp.addActionListener(e -> help());
        JMenuItem miAbout = new JMenuItem("About");
        mnHelp.setMnemonic(KeyEvent.VK_A);
        miAbout.addActionListener(e -> about());
        mnHelp.add(miHelp);
        mnHelp.addSeparator();
        mnHelp.add(miAbout);
    }

    void showUI() {
        JFrame frame = new JFrame("Client");

        // Setup statusbar
        JPanel pnStatusBar = new JPanel();
        pnStatusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));

        pnStatusBar.setPreferredSize(new Dimension(frame.getWidth(), 20));
        pnStatusBar.setLayout(new BoxLayout(pnStatusBar, BoxLayout.X_AXIS));
        pnStatusBar.add(lbStatusUser);
        pnStatusBar.add(lbStatusServer);
```

```
        setUserMenu();
        setUserList();

        frame.setLayout(new BorderLayout());
        frame.setJMenuBar(menuBar);
        frame.add(pnStatusBar, BorderLayout.SOUTH);
        frame.add(this, BorderLayout.CENTER);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(this, BorderLayout.CENTER);
        frame.pack();
        frame.setVisible(true);
    }

    // Starts a connection to the server, sets all GUI elements
    private void connect() {
        String server = JOptionPane.showInputDialog("Enter server [IP],[port]:", "127.0.0.1,5561");
        String serverValues[] = server.split(",");
        String ip = serverValues[0];
        int port = Integer.parseInt(serverValues[1]);
        cc.connect(ip, port);
        lbStatusServer.setText(", Server: IP: " + ip + ", port: " + port);
        miConnect.setEnabled(false);
        miDisconnect.setEnabled(true);
        bnSend.setEnabled(true);
        bnImage.setEnabled(true);
        mnUser.setEnabled(false);
    }

    // Ends a connection to the server, sets all GUI elements
    public void disconnect() {
        cc.disconnect();
        miConnect.setEnabled(true);
        miDisconnect.setEnabled(false);
        lbStatusServer.setText(", Disconnected");
        bnSend.setEnabled(false);
        bnImage.setEnabled(false);
        mnUser.setEnabled(true);
    }

    private void exit() {
        disconnect();
        System.exit(0);
    }

    // Add new local user
    private void addNewUser() {
        String name = "";
        while (name.equals(""))
            name = JOptionPane.showInputDialog("Enter username:");
        String imagePath = getImagePath();
        cc.addLocalUser(name, imagePath);
        setUserSelected(name);
        setUserMenu();
    }

    void selectUser(ActionEvent e) { // Select a user from the menu to connect with
        JRadioButtonMenuItem mi = (JRadioButtonMenuItem) e.getSource();
        setUserSelected(mi.getText());
    }

    void setUserMenu() { // Update the user menu (of local users) when adding a new user
        mnUser.removeAll();
        if (cc.getLocalUsers().size() > 0) {
            ButtonGroup bgUsers = new ButtonGroup();
            for (User u : cc.getLocalUsers()) {
                JRadioButtonMenuItem mi = new JRadioButtonMenuItem(u.toString());
                mi.addActionListener(e -> selectUser(e));
                bgUsers.add(mi);
                mnUser.add(mi);
                if (cc.getLocalUser() != null && mi.getText().equals(cc.getLocalUser().toString()))
                    mi.setSelected(true);
            }
            mnUser.add(new JSeparator());
        }
        mnUser.add(miNewUser);
    }

    private void setUserSelected(String sUser) { // What to do when a user has been selected from the menu
        cc.setLocalUser(sUser);
        miConnect.setEnabled(true);
        lbStatusUser.setText("User: " + sUser);
    }
```

```
    private void help() {
        JOptionPane.showMessageDialog(null, "To connect, first set username and avatar.\n" +
                "Sending messages: press Send button or ctrl-Enter.\n" +
                "Sending private messages: Select users from the user list (hold ctrl to select more than one).\n" +
                "Managing favorites: right-click on user and select Favorite.\n" +
                "Sorting user list: right click and select sort order.");
    }

    private void about() {

    }

    synchronized void setUserList() { // Update the user list (for example, when a new user connects)
        lmUsers = new UserListModel(cc.getUserlist());
        lmUsers.sort(sortBy);
        jlUsers.setModel(lmUsers);
    }


    private void toggleFavorite() { // Toggle wether a user in the user list is a favorite or not
        cc.toggleFavorite(userInList, miFavorite.getState());
        setUserList();
        jlUsers.repaint();
        userInList = null;
    }

    void sendMessage() {
        Set<User> seReceivers = null;
        int[] iaReceivers = jlUsers.getSelectedIndices(); // Get selected users from the user list
        jlUsers.clearSelection();

        if (iaReceivers.length > 0) { // See if any users were selected
            seReceivers = new HashSet<>();
            for (int u : iaReceivers) {
                seReceivers.add(lmUsers.getElementAt(u).getThinUser());
                seReceivers.add(cc.getLocalUser().getThinUser());
            }
        } else { // If not, get the members of the current chat pane
            Set<User> members = ((ChatPane) tpChatPanes.getSelectedComponent()).getMembers();
            if (members != null) {
                seReceivers = new HashSet<>();
                for (User u : members)
                    seReceivers.add(u.getThinUser());
            }
        }

            tpChatPanes.setSelectedComponent(paneController.getChatPane(seReceivers)); // Set the selected chat pane
belonging to the receiver group as active

         UserMessage userMessage = new UserMessage(cc.getLocalUser(), seReceivers, taMessage.getText(), (imageMessage
!= null ? new SerializableImage(imageMessage) : null));

        taMessage.setText("");
        imageMessage = null;
        cc.sendPacket(userMessage);
    }

    public void receiveMessage(UserMessage userMessage) {
        ChatPane pane = paneController.getChatPane(userMessage.getUsers());

        GUIHelper.putMessage(userMessage, pane);

        if (!tpChatPanes.getSelectedComponent().equals(pane))
            pane.addNotification();
    }

    public void receiveConnect(ConnectMessage connect) {
        setUserList();

        List<ChatPane> panes = paneController.getChatPanes(connect.getSender());
        for (ChatPane p : panes) { // Add a connect message to all panes the user is member of
            GUIHelper.putConnect(connect, p);
        }
    }

    public void receiveDisconnect(DisconnectMessage disconnect) {
        setUserList();

        List<ChatPane> panes = paneController.getChatPanes(disconnect.getSender());
        for (ChatPane p : panes) { // Add a disconnect message to all panes the user is member of
            GUIHelper.putDisconnect(disconnect, p);
        }
        // If it's this client that is disconnecting, set all GUI elements and end connection
        if (disconnect.getSender().equals(cc.getLocalUser()))
```

```
            disconnect();
    }

    void setMessageImage() {
        imageMessage = getImagePath();
    }

    // A method to get an image from a JFilePicker
    String getImagePath() {
        String imagePath = null;
        JFileChooser jfc = new JFileChooser();
        FileFilter imageFilter = new FileNameExtensionFilter(
                "Image files", ImageIO.getReaderFileSuffixes());
        jfc.setFileFilter(imageFilter);
        jfc.setFileSelectionMode(JFileChooser.FILES_ONLY);

        int f = jfc.showOpenDialog(null);
        if (f == JFileChooser.APPROVE_OPTION) {
            imagePath = jfc.getSelectedFile().getPath();
        }
        return imagePath;
    }

    private class SortListener implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) { // Sets the user list sort order (Name, Online, Favorite)
            sortBy = ((JRadioButtonMenuItem) e.getSource()).getText();
            lmUsers.sort(sortBy);
            jlUsers.repaint();
        }
    }

    private class EnterListener implements KeyListener {

        @Override
        public void keyPressed(KeyEvent e) { // Activate ctrl-enter to send message
            if (e.getKeyCode() == KeyEvent.VK_ENTER && e.isControlDown()) {
                e.consume();
                bnSend.doClick();
            }
        }

        @Override
        public void keyReleased(KeyEvent arg0) {
        }

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

ClientUI

```
package javachat.client;

import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.UserMessage;

// Interface if one wants to make their own UI for the chat client
public interface ClientUI {
   void receiveMessage(UserMessage userMessage);

   void receiveConnect(ConnectMessage connect);

   void receiveDisconnect(DisconnectMessage disconnect);

   void disconnect();
}
```

Controller

```
package javachat.client;

import javachat.*;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.HashSet;
import java.util.Set;

public class Controller {
   private File fiFavorites = new File("data/favorites.dat");
   private File fiUsers = new File("data/users.dat");
   private Set<User> usersOnline = new HashSet<>();
   private Set<User> usersFavorite = new HashSet<>();
   private Set<User> usersLocal = new HashSet<>();
   private User userLocal;
   private ClientUI ui;
   private ClientConnection cc;

   private Controller() {
       // Create data folder
       File foData = new File("data/");
       if (!foData.exists()) {
           foData.mkdir();
       }

       // Read list of favorites
       if (fiFavorites.exists()) {
           Object o = ChatHelper.readFile(fiFavorites);
           usersFavorite = (HashSet<User>) ChatHelper.readFile(fiFavorites);
           if (usersFavorite == null)
               usersFavorite = new HashSet<>();
       }

       // Read list of local users
       if (fiUsers.exists()) {
           System.out.println("Reading filed users.dat...");
           usersLocal = (HashSet<User>) ChatHelper.readFile(fiUsers);
           if (usersLocal == null)
               usersLocal = new HashSet<>();
       }
   }

   public void setUI(ClientUI ui) {
       this.ui = ui;
   }

   public void addLocalUser(String name, String imagePath) {
       SerializableImage ii = null;
       if (imagePath == null) { // If no image is set, set image to gubbe.jpg
           try {
               BufferedImage bIm = ImageIO.read(new URL("https://i.imgur.com/B87P6eN.jpg"));
               ByteArrayOutputStream baos = new ByteArrayOutputStream();
               ImageIO.write(bIm, "png", baos);
               baos.flush();
               ii = new SerializableImage(baos.toByteArray());
           } catch (IOException e) {
               System.out.println("Image error: [" + e.toString() + "]");
           }
       } else {
           ii = new SerializableImage(imagePath);
       }
       User newUser = new User(name, ii);
       System.out.println("Adding user " + newUser);
       if (usersLocal.contains(newUser)) {
           System.out.println("User exists");
           for (User u : usersLocal)
               if (u.equals(newUser))
                   u.setAvatar(newUser.getAvatar()); // If user exists, update avatar
       } else {
           System.out.println("User doesn't exist");
           usersLocal.add(newUser);
       }
       ChatHelper.writeToFile(usersLocal, fiUsers);
   }

   public Set<User> getLocalUsers() {
```

```
        return usersLocal;
    }

    // Select which user to log in with
    public void setLocalUser(String name) {
        for (User u : usersLocal)
            if (name.equals(u.toString()))
                userLocal = u;
    }

    public User getLocalUser() {
        return userLocal;
    }

    // Get list of online users, and offline favorites
    public Set<User> getUserlist() {
        Set<User> users = new HashSet<>();
        for (User u : usersOnline)
            if (usersFavorite.contains(u))
                u.setFavorite(true);
        System.out.println("Online size: " + usersOnline.size() + ", " + usersOnline);
        users.addAll(usersOnline);
        System.out.println("Favorite size: " + usersFavorite.size() + ", " + usersFavorite);
        users.addAll(usersFavorite);

        return users;
    }

    public void connect(String ip, int port) {
        cc = new ClientConnection(this, ip, port);
        cc.connect(new ConnectMessage(userLocal));
    }

    public void disconnect() {
        if (cc != null)
            cc.sendPacket(new DisconnectMessage(userLocal));
    }

    public void sendPacket(Message message) {
        cc.sendPacket(message);
    }

    public void receivePacket(Message message) {
        if (message instanceof UserMessage)
            ui.receiveMessage((UserMessage) message);
        else if (message instanceof ConnectMessage)
            receiveConnect(((ConnectMessage) message));
        else if (message instanceof DisconnectMessage)
            receiveDisconnect((DisconnectMessage) message);
    }

    // Receiving a Message of a new connected user
    private void receiveConnect(ConnectMessage connect) {
        if (connect.getSender().equals(userLocal)) { // If it's my own connections message, add all users
            usersOnline.addAll(connect.getUsers());
        } else {
            usersOnline.add(connect.getSender()); // Otherwise, just add the latest user
            System.out.println("Server/Client list match: " + usersOnline.equals(connect.getUsers()));
        }
        System.out.println("Client: " + connect.getSender() + " connected");
        ui.receiveConnect(connect);
    }

    // Receiving a Message of a disconnected user
    private void receiveDisconnect(DisconnectMessage disconnect) {
            if (disconnect.getSender().equals(userLocal)) { // If it's my own disconnect message echoing, shut down
connection
            usersOnline.clear();
            cc = null;
            System.out.println("Client disconnected.");
        } else {
            usersOnline.remove(disconnect.getSender());
            System.out.println("Client: " + disconnect.getSender() + " disconnected");
        }
        ui.receiveDisconnect(disconnect);
    }

    public void toggleFavorite(User user, boolean state) {
        if (true) { //!user.equals(userLocal)) { For testing purposes, one can favorite themselves atm
            user.setFavorite(state);
            System.out.println("User is favorite: " + user.isFavorite());
            if (!user.isFavorite()) {
                usersFavorite.remove(user);
            } else {
```

```
                if (true) { // !user.equals(userLocal)) {
                    User fav = new User(user.toString(), user.getAvatar());
                    fav.setFavorite(true);
                    fav.setOnline(false);
                    usersFavorite.add(fav);
                }
            }
        }
        ChatHelper.writeToFile(usersFavorite, fiFavorites);
    }

    public static void main(String[] args) {
        //System.setProperty("sun.java2d.uiScale", "1.0");
        System.setProperty("apple.laf.useScreenMenuBar", "true");
        Controller cc = new Controller();
        ClientGUI gui = new ClientGUI(cc);
        cc.setUI(gui);
        gui.showUI();
    }
}
```

GUIHelper

```
package javachat.client;

import javachat.ChatHelper;
import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.UserMessage;

import javax.swing.text.Style;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyleContext;
import java.awt.*;
import java.text.SimpleDateFormat;

public class GUIHelper {
   private static StyleContext sc = new StyleContext();
   private static SimpleDateFormat dateFormatter = new SimpleDateFormat("HH:mm");

   private static void loadStyles() {
       Style cRed = sc.addStyle("cRed", null);
       Style cBlue = sc.addStyle("cBlue", null);
       Style cGreen = sc.addStyle("cGreen", null);
       Style cBlack = sc.addStyle("cBlack", null);
       Style fMono = sc.addStyle("fMono", null);
       cRed.addAttribute(StyleConstants.Foreground, Color.RED);
       cBlue.addAttribute(StyleConstants.Foreground, Color.BLUE);
       cGreen.addAttribute(StyleConstants.Foreground, new Color(0, 180, 0));
       cBlack.addAttribute(StyleConstants.Foreground, Color.BLACK);
       fMono.addAttribute(StyleConstants.FontFamily, "Monospaced");
   }

   private static Style getStyle(String style) {
       return sc.getStyle(style);
   }

   // Add a text to a Chatpane with proper formatting
   public static void putMessage(UserMessage userMessage, ChatPane pane) {
       loadStyles();

       if (userMessage.getImage() != null) {
           double imgWidth = userMessage.getImage().getIconWidth(),
                   imgHeight = userMessage.getImage().getIconHeight(),
                   paneWidth = pane.getSize().getWidth(),
                   paneHeight = pane.getSize().getHeight(), ratio;

           if (imgWidth > paneWidth || imgHeight > paneHeight) { // If image is wider/higher than Chatpane, resize
               if (imgWidth / imgHeight > paneWidth / paneHeight)
                   ratio = paneWidth / imgWidth;
               else
                   ratio = paneHeight / imgHeight;
               imgWidth *= ratio;
               imgHeight *= ratio;
                    pane.insertImage(ChatHelper.getScaledImage(userMessage.getImage().getImageIcon(), (int) imgWidth,
(int) imgHeight));
           } else
               pane.insertImage(userMessage.getImage());

           pane.insertText("\n", getStyle("cBlack"));
       }

       pane.insertText("[" + dateFormatter.format(userMessage.getDelivered()) + "] ", getStyle("black"));
       pane.insertText("<", getStyle("cBlue"));
       pane.insertText(userMessage.getSender().toString(), getStyle("cRed"));
       pane.insertText("> ", getStyle("cBlue"));
       pane.insertText(userMessage.getContent() + "\n", getStyle("cBlack"));
       pane.jumpToBottom();
   }

   // Add a connect message to Chatpane with proper formatting
   public static void putConnect(ConnectMessage connect, ChatPane pane) {
       loadStyles();
           pane.insertText("[" + dateFormatter.format(connect.getDelivered()) + "] " + connect.getSender() + " has
connected\n", getStyle("cGreen"));
       pane.jumpToBottom();
   }

   // Add a disconnect messsage to Chatpane with proper formatting
   public static void putDisconnect(DisconnectMessage disconnect, ChatPane pane) {
       loadStyles();
        pane.insertText("[" + dateFormatter.format(disconnect.getDelivered()) + "] " + disconnect.getSender() + " has
disconnected\n", getStyle("cRed"));
       pane.jumpToBottom();
```

```
    }
}
```

Källkod för gemensamma klasser i klient och server

ChatHelper

```
package javachat.client;

import javachat.ChatHelper;
import javachat.ConnectMessage;
import javachat.DisconnectMessage;
import javachat.UserMessage;

import javax.swing.text.Style;
import javax.swing.text.StyleConstants;
import javax.swing.text.StyleContext;
import java.awt.*;
import java.text.SimpleDateFormat;

public class GUIHelper {
   private static StyleContext sc = new StyleContext();
   private static SimpleDateFormat dateFormatter = new SimpleDateFormat("HH:mm");

   private static void loadStyles() {
       Style cRed = sc.addStyle("cRed", null);
       Style cBlue = sc.addStyle("cBlue", null);
       Style cGreen = sc.addStyle("cGreen", null);
       Style cBlack = sc.addStyle("cBlack", null);
       Style fMono = sc.addStyle("fMono", null);
       cRed.addAttribute(StyleConstants.Foreground, Color.RED);
       cBlue.addAttribute(StyleConstants.Foreground, Color.BLUE);
       cGreen.addAttribute(StyleConstants.Foreground, new Color(0, 180, 0));
       cBlack.addAttribute(StyleConstants.Foreground, Color.BLACK);
       fMono.addAttribute(StyleConstants.FontFamily, "Monospaced");
   }

   private static Style getStyle(String style) {
       return sc.getStyle(style);
   }

   // Add a text to a Chatpane with proper formatting
   public static void putMessage(UserMessage userMessage, ChatPane pane) {
       loadStyles();

       if (userMessage.getImage() != null) {
           double imgWidth = userMessage.getImage().getIconWidth(),
                   imgHeight = userMessage.getImage().getIconHeight(),
                   paneWidth = pane.getSize().getWidth(),
                   paneHeight = pane.getSize().getHeight(), ratio;

           if (imgWidth > paneWidth || imgHeight > paneHeight) { // If image is wider/higher than Chatpane, resize
               if (imgWidth / imgHeight > paneWidth / paneHeight)
                   ratio = paneWidth / imgWidth;
               else
                   ratio = paneHeight / imgHeight;
               imgWidth *= ratio;
               imgHeight *= ratio;
                   pane.insertImage(ChatHelper.getScaledImage(userMessage.getImage().getImageIcon(), (int) imgWidth,
(int) imgHeight));
           } else
               pane.insertImage(userMessage.getImage());

           pane.insertText("\n", getStyle("cBlack"));
       }

       pane.insertText("[" + dateFormatter.format(userMessage.getDelivered()) + "] ", getStyle("black"));
       pane.insertText("<", getStyle("cBlue"));
       pane.insertText(userMessage.getSender().toString(), getStyle("cRed"));
       pane.insertText("> ", getStyle("cBlue"));
       pane.insertText(userMessage.getContent() + "\n", getStyle("cBlack"));
       pane.jumpToBottom();
   }

   // Add a connect message to Chatpane with proper formatting
   public static void putConnect(ConnectMessage connect, ChatPane pane) {
       loadStyles();
           pane.insertText("[" + dateFormatter.format(connect.getDelivered()) + "] " + connect.getSender() + " has
connected\n", getStyle("cGreen"));
       pane.jumpToBottom();
```

```
    }

    // Add a disconnect messsage to Chatpane with proper formatting
    public static void putDisconnect(DisconnectMessage disconnect, ChatPane pane) {
        loadStyles();
         pane.insertText("[" + dateFormatter.format(disconnect.getDelivered()) + "] " + disconnect.getSender() + " has
disconnected\n", getStyle("cRed"));
        pane.jumpToBottom();
    }
}
```

ConnectMessage

```
package javachat;

import java.net.Socket;

public class ConnectMessage extends Message {
  private transient Socket socket;

  public ConnectMessage(User sender) {
     super(sender);
     super.getSender().setOnline(true);
  }

  public void setSocket(Socket socket) {
     this.socket = socket;
  }

  public Socket getSocket() {
     return socket;
  }

}
```

DisconnectMessage

```
package javachat;

public class DisconnectMessage extends Message {

  public DisconnectMessage(User sender) {
     super(sender.getThinUser());
  }
}
```

Message

```
package javachat;

import java.io.Serializable;
import java.util.Date;
import java.util.Set;

public abstract class Message implements Serializable {

  private static final long serialVersionUID = -8375731807804305953L;
  private User sender;
  protected Set<User> users;
  private Date arrived;
  private Date delivered;

  public Message(User sender) {
     this.sender = sender;
  }

  public User getSender() {
     return sender;
  }

  public Set<User> getUsers() {
     return users;
  }

  public void setUsers(Set<User> users) {
     this.users = users;
  }

  public void setArrived(Date date) {
     arrived = date;
  }

  public Date getArrived() {
     return arrived;
  }

  public void setDelivered(Date date) {
     delivered = date;
  }

  public Date getDelivered() {
     return delivered;
  }
}
```

SerializableImage

```
package javachat;

import javax.swing.*;
import java.awt.*;
import java.io.File;
import java.io.Serializable;
import java.nio.file.Files;

/**
* SerializableImage wraps ImageIcon so that it can be serialized between
* different versions of Java.
*
* @author Anders Bergh
*/
public class SerializableImage implements Serializable, Icon {
   private byte[] imageData;
   private transient ImageIcon imageIcon;

   /**
    * Constructs a SerializableImage from the contents in the specified file.
    *
    * @param path Path to image to load.
    */
   public SerializableImage(String path) {
       try {
           imageData = Files.readAllBytes(new File(path).toPath());
           getImageIcon();
       } catch (Exception e) {
           e.printStackTrace();
       }
   }

   /**
    * Constructs a SerializableImage from a byte array of image data.
    *
    * @param imageData The image to load.
    */
   public SerializableImage(byte[] imageData) {
       this.imageData = imageData;
       getImageIcon();
   }

   /**
    * Returns a Swing ImageIcon from the encapsulated image data.
    *
    * @return An instance of ImageIcon.
    */
   public ImageIcon getImageIcon() {
       if (imageData != null && imageIcon == null) {
           imageIcon = new ImageIcon(imageData);
       }
       return imageIcon;
   }

   @Override
   public void paintIcon(Component c, Graphics g, int x, int y) {
       imageIcon.paintIcon(c, g, x, y);
   }

   @Override
   public int getIconWidth() {
       if (imageIcon == null)
           getImageIcon();
       return imageIcon.getIconWidth();
   }

   @Override
   public int getIconHeight() {
       if (imageIcon == null)
           getImageIcon();
       return imageIcon.getIconHeight();
   }
}
```

User

```
package javachat;

import java.io.Serializable;

public class User implements Serializable {
  private static final long serialVersionUID = -8518522418186109463L;
  private String name;
  private SerializableImage avatar;
  private boolean online = false;
  private boolean favorite = false;

  public User(String name, SerializableImage avatar) {
     this.name = name;
     this.avatar = avatar;
  }

  public String getName() {
     return name;
  }

  public void setAvatar(SerializableImage avatar) {
     this.avatar = avatar;
  }

  public SerializableImage getAvatar() {
     return avatar;
  }

  public void setOnline(boolean online) {
     this.online = online;
  }

  public boolean isOnline() {
     return online;
  }

  public void setFavorite(boolean favorite) {
     this.favorite = favorite;
  }

  public boolean isFavorite() {
     return favorite;
  }

  @Override
  public String toString() {
     return name;
  }

  public int hashCode() {
     return name.hashCode();
  }

  @Override
  public boolean equals(Object o) {
     // self check
     if (this == o) {
        return true;
     }

     if (!(o instanceof User)) {
        return false;
     }

     User u = (User) o;

     // field comparison
     return u.toString().equals(name);
  }

  public User getThinUser() {
     User u = new User(name, null);
     u.setOnline(true);
     return u;
  }
}
```

UserCellRenderer

```
package javachat;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;

import javax.swing.*;
import javax.swing.border.BevelBorder;

// Class for showing how to render a user in a JList (name, online, favorite, avatar)
public class UserCellRenderer extends JPanel implements ListCellRenderer<User> {

   @Override
      public Component getListCellRendererComponent(JList<? extends User> list, User user, int index, boolean
isSelected, boolean cellHasFocus) {
       removeAll();
       setSize(200, 100);
       JLabel lbStatus = new JLabel();
       lbStatus.setPreferredSize(new Dimension(20, 20));
       if (user.isOnline()) {
           lbStatus.setOpaque(true);
           lbStatus.setBackground(Color.GREEN);
       }
       if (user.isFavorite())
           lbStatus.setText(Character.toString((char) 9733));
       add(lbStatus);
       if (isSelected)
           setBorder(BorderFactory.createBevelBorder(BevelBorder.LOWERED));
       else
           setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
       JLabel lbName = new JLabel(user.toString());
       lbName.setPreferredSize(new Dimension(80, 100));
       add(lbName);
       JLabel lbAvatar = new JLabel();
       lbAvatar.setPreferredSize(new Dimension(100, 100));
       lbAvatar.setIcon(ChatHelper.getScaledImage(user.getAvatar().getImageIcon(), 100, 100));
       add(lbAvatar);

       return this;
   }
}
```

UserListModel

```
package javachat;

import javachat.User;

import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;

import javax.swing.ListModel;
import javax.swing.event.ListDataListener;

// Class for helping organize user list after name, online, or favorite
public class UserListModel implements ListModel<User> {
  private List<User> users = new LinkedList<User>();

  public UserListModel(Set<User> userList) {
     addList(userList);
  }

  public void addList(Set<User> userList) {
     for (User u: userList)
        users.add(u);
  }

  @Override
  public void addListDataListener(ListDataListener arg0) {
  }

  @Override
  public User getElementAt(int index) {
     return users.get(index);
  }

  @Override
  public int getSize() {
     return users.size();
  }

  @Override
  public void removeListDataListener(ListDataListener arg0) {
  }

  public void sort(String button) {
     if (button.equals("Name"))
        users.sort(new SortByName());
     else if (button.equals("Favorite"))
        users.sort(new SortByFavorite());
     else if (button.equals("Online"))
        users.sort(new SortByOnline());
  }

  private class SortByName implements Comparator<User> {

     @Override
     public int compare(User u1, User u2) {
        return u1.toString().compareTo(u2.toString());
     }
  }

  private class SortByOnline implements Comparator<User> {

     @Override
     public int compare(User u1, User u2) {
        if (u1.isOnline() == u2.isOnline())
           return u1.toString().compareTo(u2.toString());
        else if (u1.isOnline() && !u2.isOnline())
           return -1;
        else
           return 1;
     }
  }

  private class SortByFavorite implements Comparator<User> {

     @Override
     public int compare(User u1, User u2) {
        if (u1.isFavorite() == u2.isFavorite())
           return u1.toString().compareTo(u2.toString());
        else if (u1.isFavorite() && !u2.isFavorite())
           return -1;
```

```
        else
            return 1;
    }
  }
}
```

UserMessage

```
package javachat;

import java.util.Set;

public class UserMessage extends Message {
  private String content;
  private SerializableImage image;

  public UserMessage(User sender, Set<User> receivers, String content, SerializableImage image) {
     super(sender);
     super.users = receivers;
     this.content = content;
     this.image = image;
  }

  public String getContent() {
     return content;
  }

  public SerializableImage getImage() {
     return image;
  }
}
```