Gruppuppgift DA343A VT18

Grupp 8

Medlemmar
Martin Gyllström
Matthias Svensson Falk
Elin Olsson
Mikael Lindfors
Oskar Engström Magnusson
Eric Grevillius

Innehållsförteckning

Gruppuppgift DA343A VT18 **0**

Innehållsförteckning **1**

Arbetsbeskrivning **3**

Instruktioner för programstart **4**

Systembeskrivning **4**

Klassdiagram **5**

Klassdiagram client 5

Klassdiagram Server 6

Sekvensdiagram **7**

1. Logga in klient (serversida) 7

2. Start av klient 9

3. Skicka meddelande(serversida) 9

4. Ta emot uppdaterad mottagarlista(klientsida) 11

5. Visning av log mellan två tidpunkter 12

6. Stänger klient 12

Källkod server **13**

ChatServer 13

ClientListener 20

LogUI 21

RunOnThreadN 22

ServerController 25

ServerUI 27

StartServer 32

UnsentMessages 33

Källkod klient **35**

ChatClient 35

ClientController 38

LoginUI 43

MessageUI 46

ServerListener 51

StartClient 51

Källkod för gemensamma klasser i klient och server **52**
LogReader 52
LogWriter 56
Message 58
User 62
UserList 64

Arbetsbeskrivning

Elin Olsson
Arbetat med Oskar(parprogramering) med att skriva klasserna User och Message. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Skriftlig dokumentation. Ansvar för sekvensdiagram nr 1

Oskar Engström Magnusson
Arbetat med Elin (parprogramering) med att skriva klasserna User och Message. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Ansvar för sekvensdiagram nr 3(Skicka meddelande)

Eric Grevilius
Arbetat med design av UI av både chatt och inloggning. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Ansvar för sekvensdiagram nr 4 “ta emot en uppdaterad mottagarlista”

Matthias Svensson Falk
Har arbetat med UI på server sidan. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Ansvar för sekvensdiagram nr 6 “close client”

Martin Gyllström
Arbetat med Mikael (parprogrammering) med Server och Klient klasser. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Ansvar för sekvensdiagram nr 2, Start av klient.

Mikael Lindfors
Arbetat med Martin (parprogrammering) med Server och Klient klasser. Vi har arbetat mycket i grupp där alla varit närvarande och aktiva vid alla grupptillfällen. Ansvar för sekvensdiagram nr 5

Instruktioner för programstart

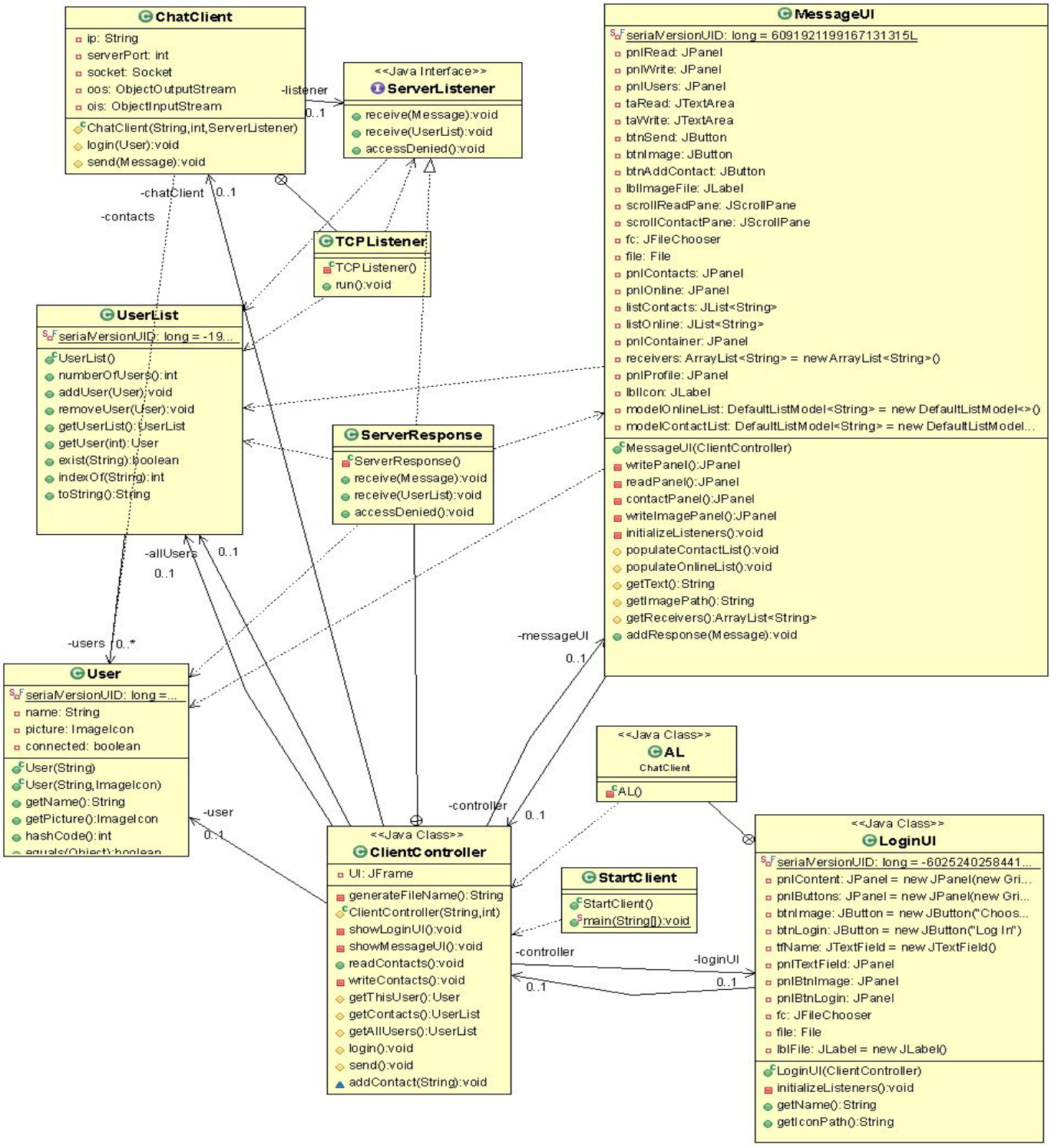
Klienten startas med StartClient och servern med StartServer, där finns main metoderna. JPG filen “no-icon” ska infogas från images till Eclipse om ingen bild av användaren väljs. Alla java-filer till klienten ligger i ChatClient, server i ChatServer och gemensamma java-filer ligger i resources. Alla bifogade java-filer ska placeras i ett paket som heter pg.

Systembeskrivning

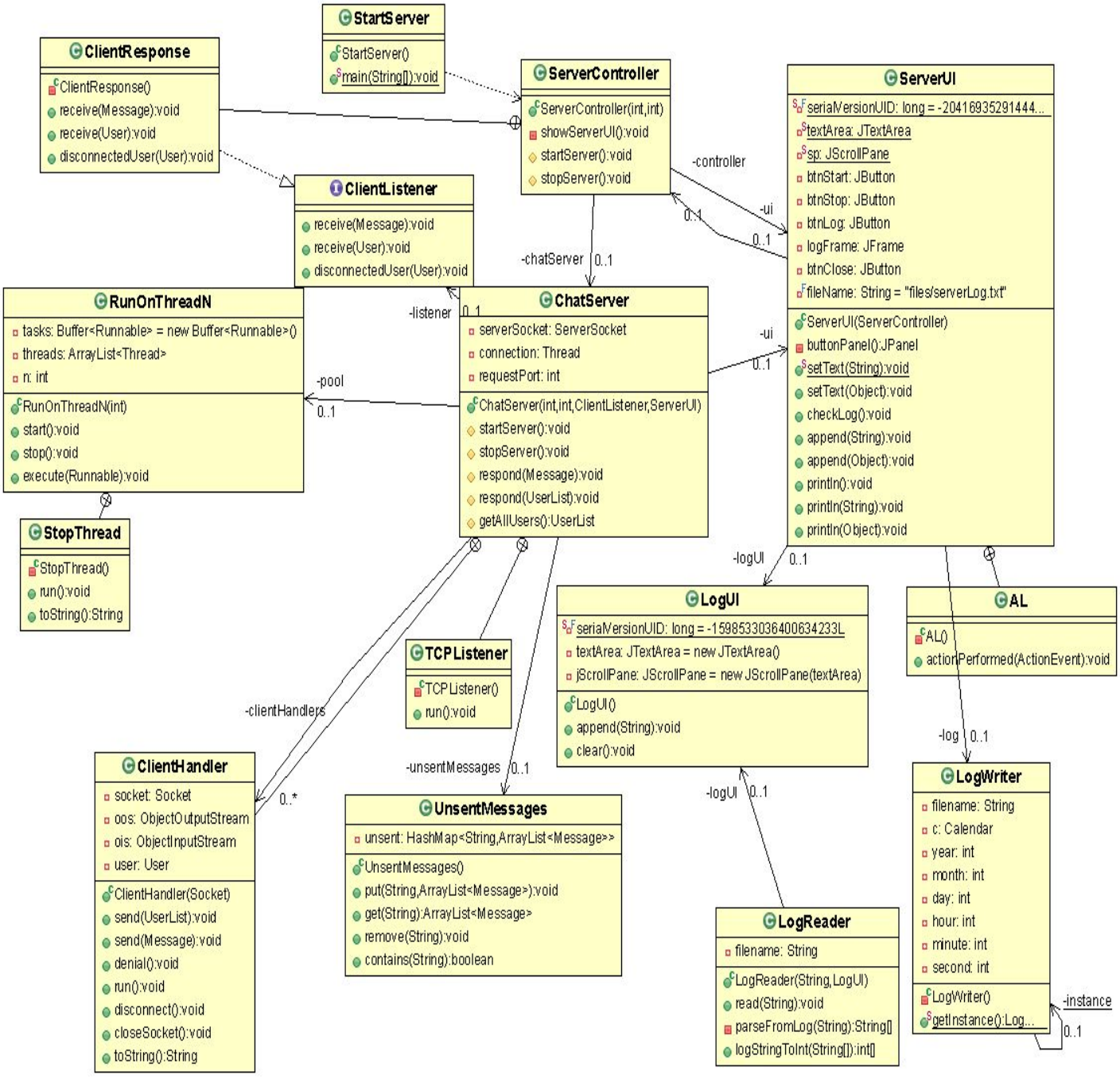
En användare kan logga in, genom att skriva ett valfritt namn och välja en bild som används som profilbild, om användaren inte valt bild så för den automatisk en standardbild. Användaren kan själv bestämma till vilka personer den vill skicka meddelandet till, antingen specifik person, alla inloggade, eller om användaren lagt till personen som kontakt. Detta sker genom att klicka på checkboxen jämte respektive persons bild. Man ska kunna skicka meddelanden till personer som inte är online, och de får senare meddelandet när de loggat in.

Klassdiagram

Klassdiagram client



Klassdiagram Server

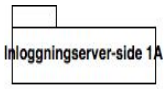


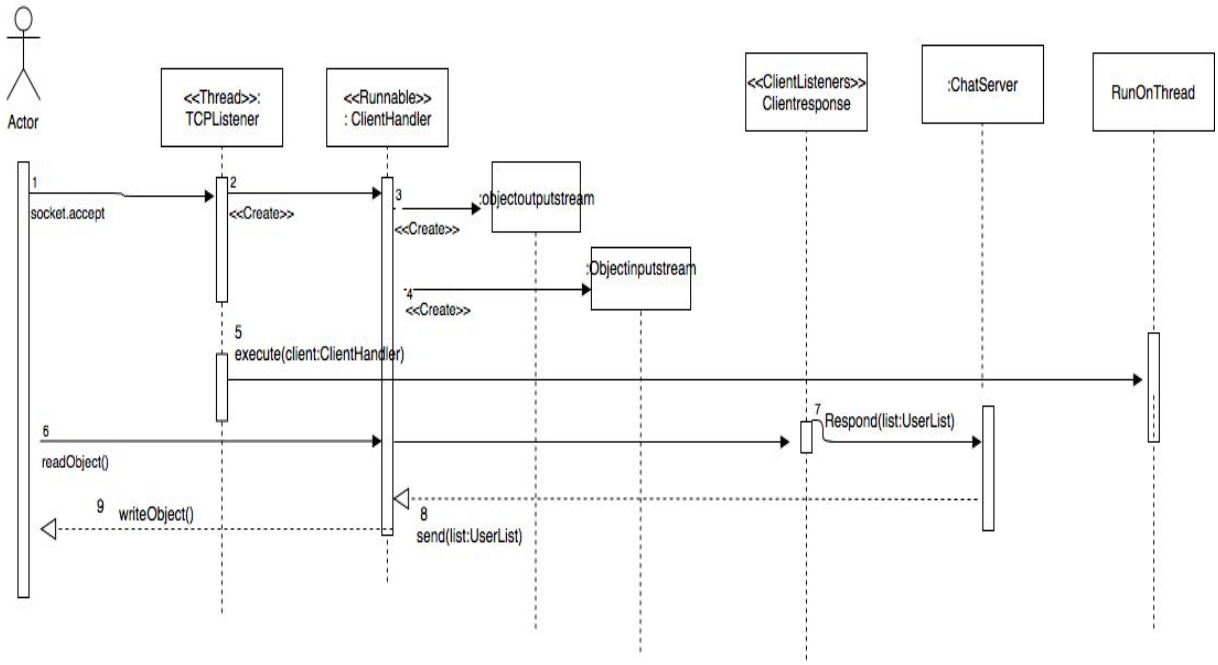
Sekvensdiagram

1. Logga in klient (serversida)

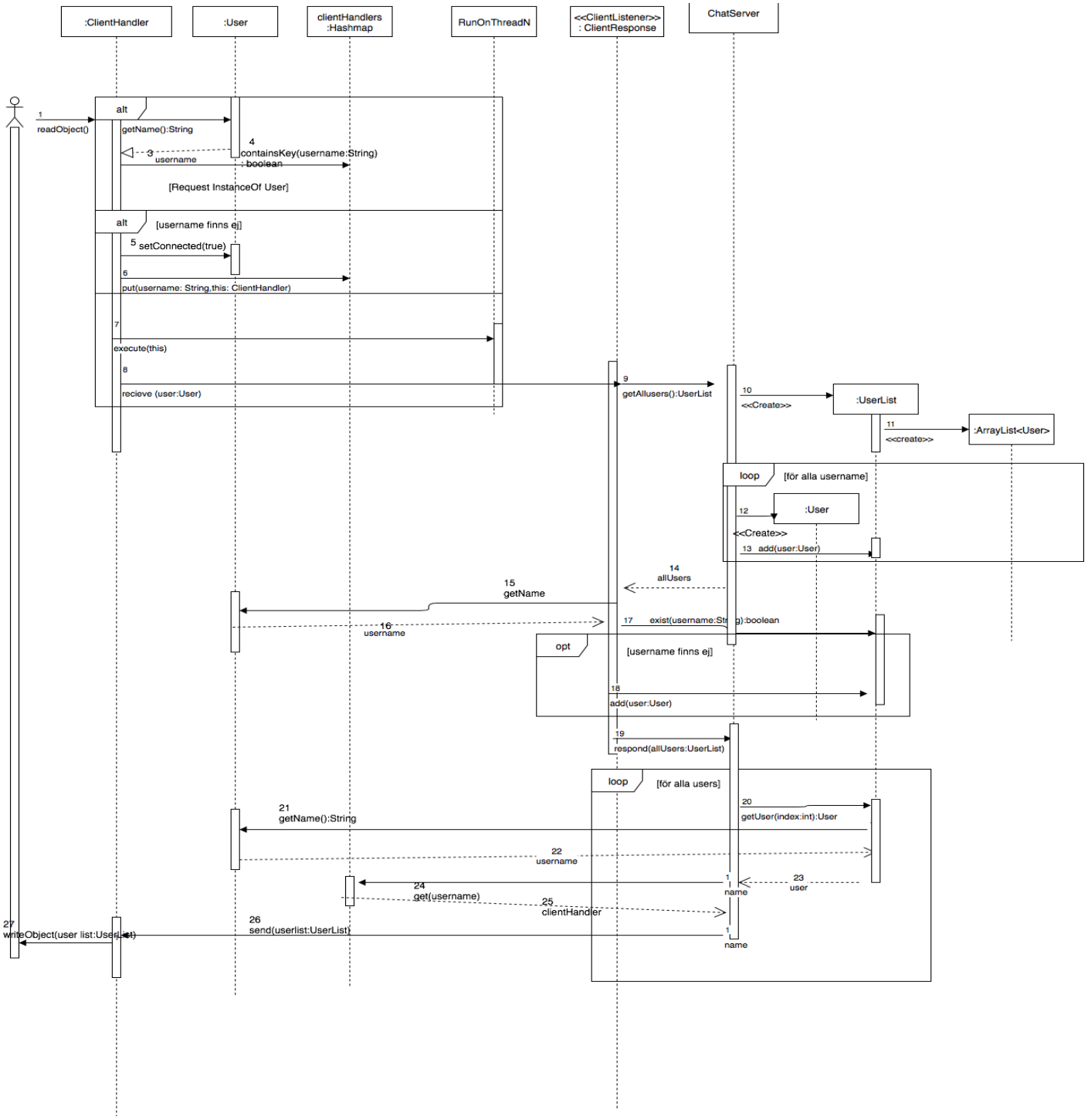
Ansvarig: Elin Olsson

1a



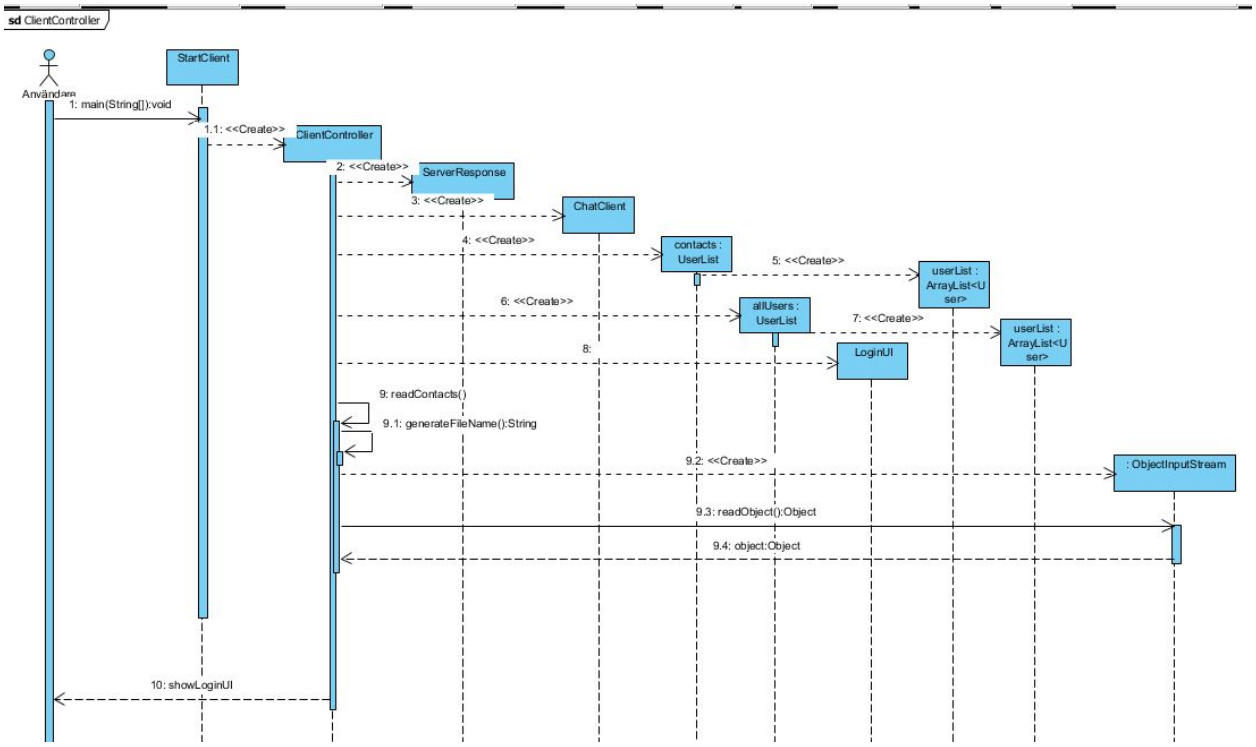


1b


:ClientHandler
:User
clientHandlers :Hashmap
RunOnThreadN
<<ClientListener>> : ClientResponse
ChatServer
alt
1 readObject()
getName():String
3 username
4 containsKey(username:String) : boolean
[Request InstanceOf User]
alt
[username finns ej]
5 setConnected(true)
6 put(username: String,this: ClientHandler)
7 execute(this)
8 recieve (user:User)
9 getAllusers():UserList
10 <<Create>>
:UserList
11 <<create>>
:ArrayList<User>
loop
[för alla username]
12
:User
<<Create>>
13 add(user:User)
14 allUsers
15 getName
16 username
17 exist(username:String):boolean
opt
[username finns ej]
18 add(user:User)
19 respond(allUsers:UserList)
loop
[för alla users]
20 getUser(index:int):User
21 getName():String
22 username
23 user
24 get(username)
25 clientHandler
26 send(userlist:UserList)
27 writeObject(user list:UserList)
name
name

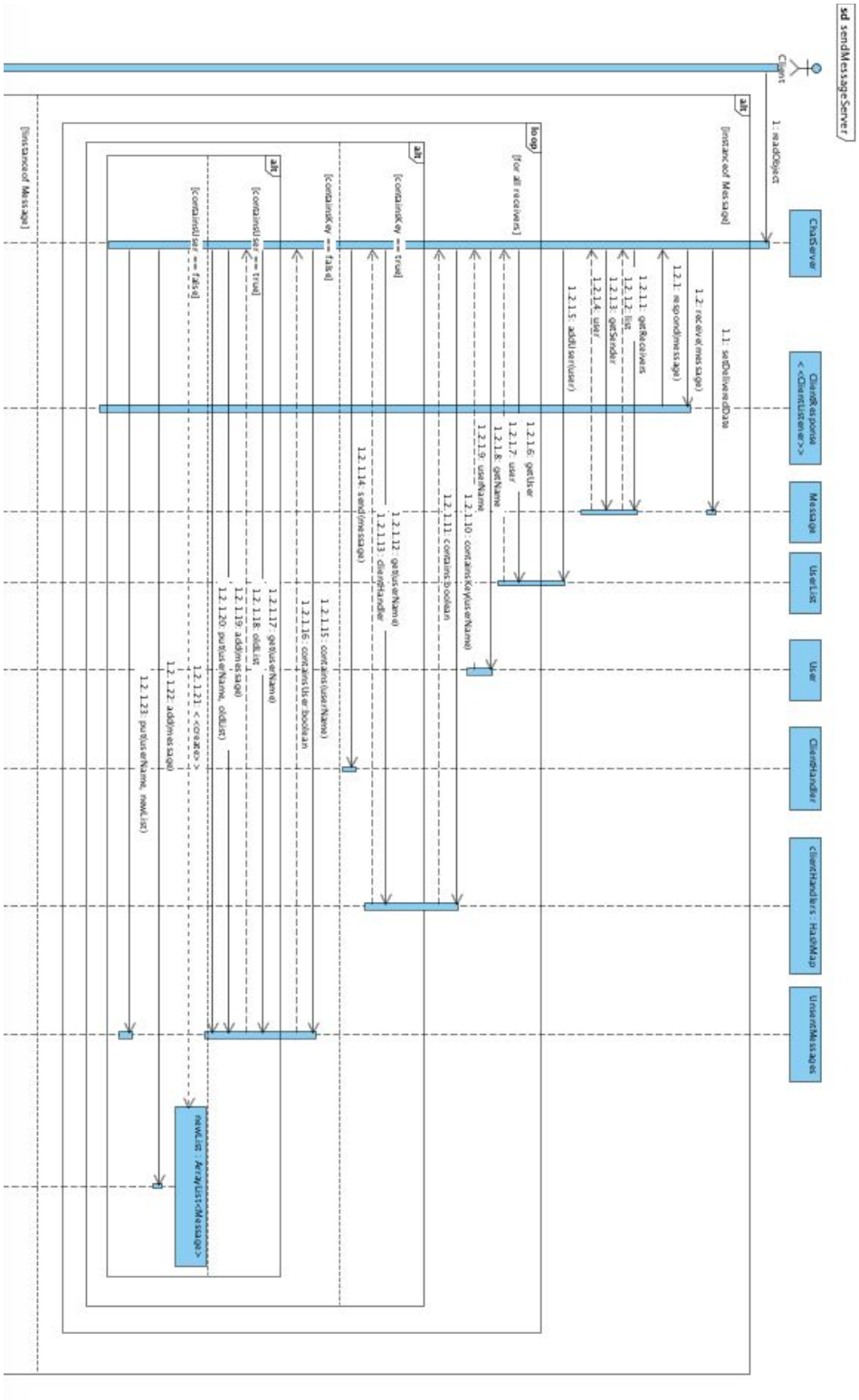
2. Start av klient

Ansvarig: Martin Gyllström



3. Skicka meddelande(serversida)

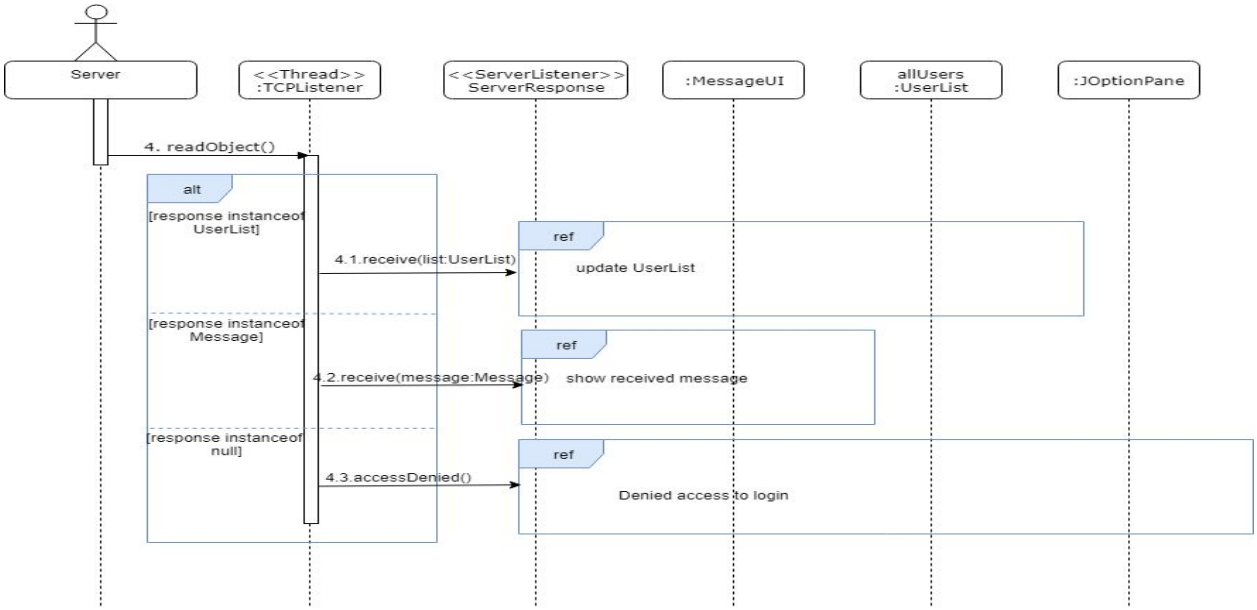
Ansvarig: Oskar Engström Magnusson



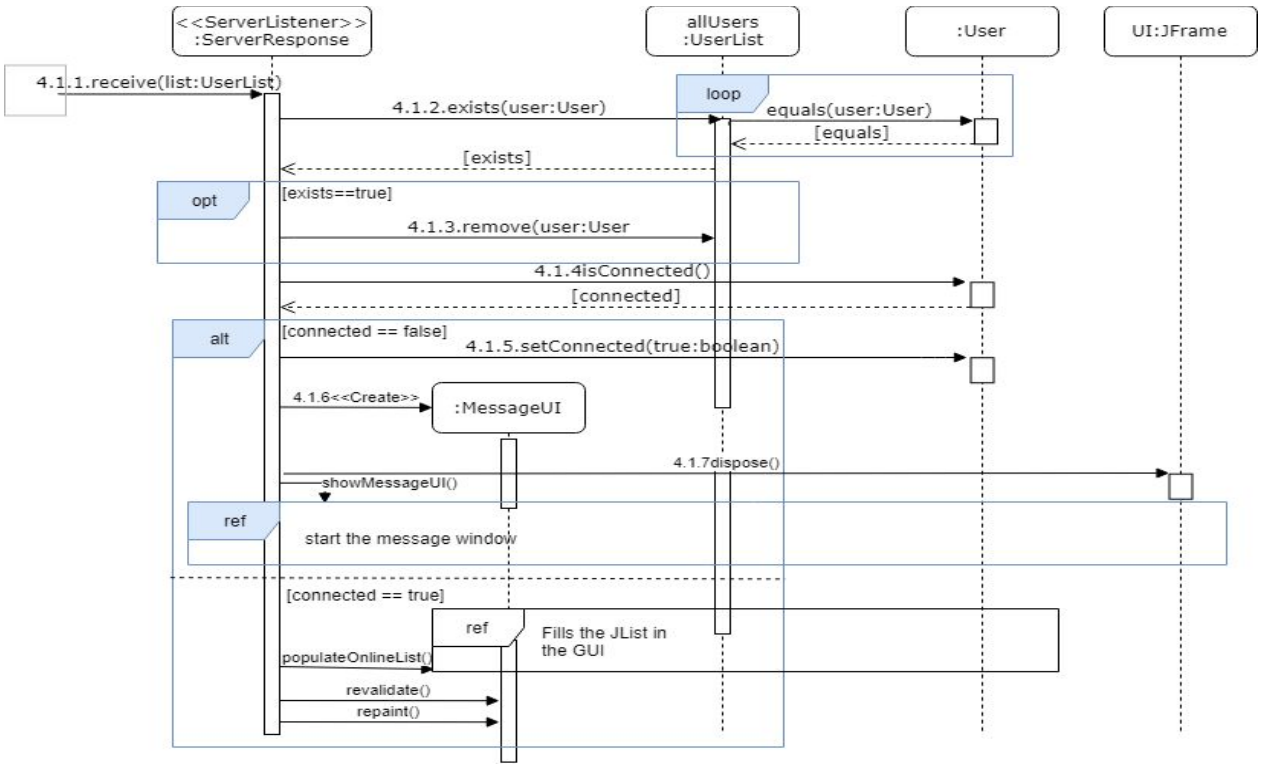
4. Ta emot uppdaterad mottagarlista(klientsida)

Ansvarig: Eric Grevillius

Ta emot objekt:

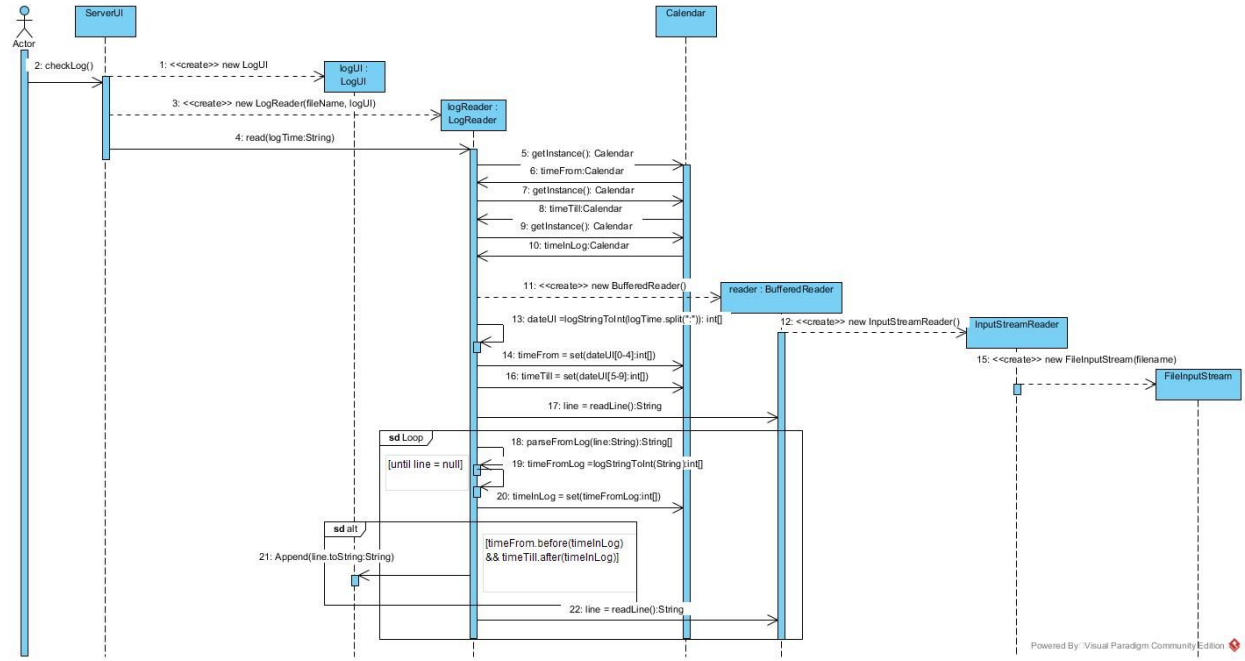


Uppdatera användarlista:



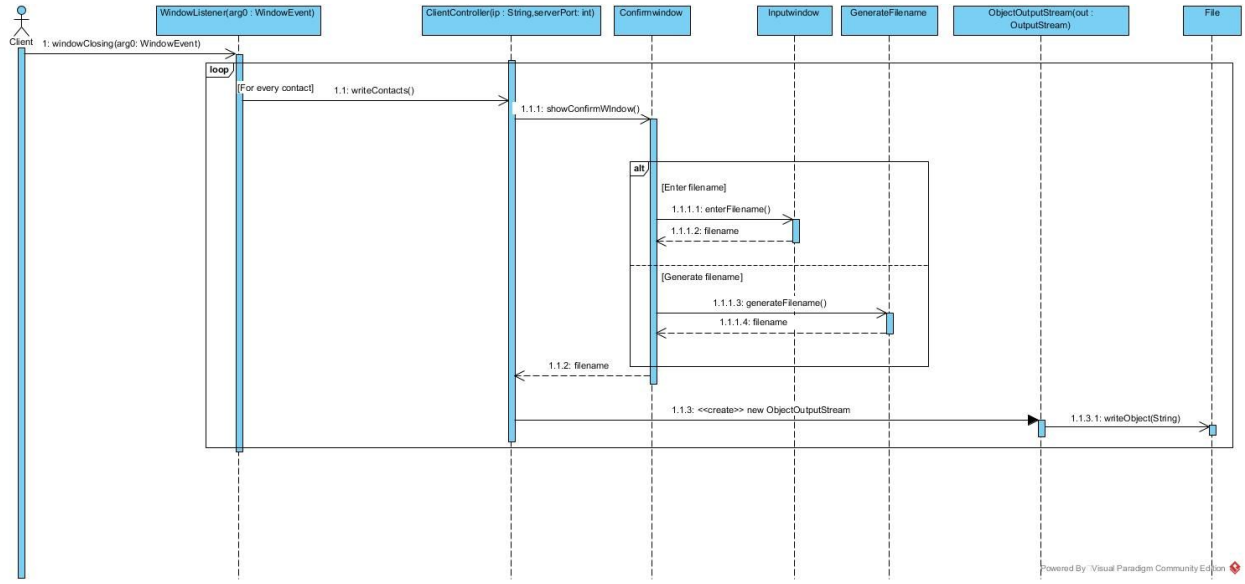
5. Visning av log mellan två tidpunkter

Ansvarig: Mikael Lindfors



6. Stänger klient

Ansvarig: Matthias Svensson Falk



Källkod server

ChatServer

```
package ChatServer;

import java.io.*;
import java.net.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

import resources.*;

/**
 * Class that handles the functionality in the server
 * @author Oskar Engsström Magnusson
 *
 */
public class ChatServer {
	private ServerUI ui;
	private ServerSocket serverSocket;
	private ClientListener listener;
	private HashMap<String, ClientHandler> clientHandlers;
	private UnsentMessages unsentMessages;
	private RunOnThreadN pool;
	private Thread connection;
	private int requestPort;

	/**
	 * Constructor for creating the server
	 * @param requestPort the port the server runs through
	 * @param nbrOfThreads number of threads in the thread-pool
	 * @param listener the ClientListener
	 * @param ui the server ui
	 */
	public ChatServer(int requestPort, int nbrOfThreads, ClientListener listener,
ServerUI ui) {
		this.listener = listener;
		this.unsentMessages = new UnsentMessages();
```

```
                this.clientHandlers = new HashMap<String, ClientHandler>();
                this.pool = new RunOnThreadN(nbrOfThreads);
                this.requestPort = requestPort;
                this.ui = ui;
        }

        /**
         * Method for starting the server by creating a ServerSocket and a TCPListener
         */
        protected void startServer() {
                try {
                        serverSocket = new ServerSocket(requestPort);
                        connection = new TCPListener();
                        connection.start();
                        pool.start();
                        ui.println("Starting Server: " +
serverSocket.getInetAddress().getHostAddress() + ":"
                                        + serverSocket.getLocalPort());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Method for stopping the server
         */
        protected void stopServer() {
                try {
                        connection.interrupt();
                        serverSocket.close();
                        pool.stop();
                        clientHandlers.clear();
                        ui.println("Server closing");
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Method that receives a message and sends it to all receivers. If the user is not
online,
         * the message is added to "Unsent messages"
         * @param message a message object
```

```
	 */
	protected void respond(Message message) {
		UserList list = message.getReceivers();
		list.addUser(message.getSender());
		for (int i = 0; i < list.numberOfUsers(); i++) {
			if(clientHandlers.containsKey(list.getUser(i).getName())){

clientHandlers.get(list.getUser(i).getName()).send(message);
			} else {
				if (unsentMessages.contains(list.getUser(i).getName())) {
					ArrayList<Message> oldList =
unsentMessages.get(list.getUser(i).getName());
					oldList.add(message);
					unsentMessages.put(list.getUser(i).getName(),
oldList);
				} else {
					ArrayList<Message> newList = new
ArrayList<Message>();
					newList.add(message);
					unsentMessages.put(list.getUser(i).getName(),
newList);
				}
			}
		}

		/**
		 * Method that receives a UserList and sends it to all users in the list
		 * @param listToSend a UserList
		 */
		protected void respond(UserList listToSend) {
			for (int i = 0; i < listToSend.numberOfUsers(); i++) {

clientHandlers.get(listToSend.getUser(i).getName()).send(listToSend);
			}
		}

		/**
		 * Method that returns a list of all users
		 * @return list of all users
		 */
		protected UserList getAllUsers() {
			UserList allUsers = new UserList();
```

```
                Iterator<String> iter = clientHandlers.keySet().iterator();
                while (iter.hasNext()) {
                        allUsers.addUser(new User(iter.next(), null));
                }
                return allUsers;
        }

        /**
         * Inner class which is listening for users to connect and adds a thread for each
user
         * @author Oskar Engström Magnusson
         *
         */
        private class TCPListener extends Thread {
                public void run() {
                        while (!Thread.interrupted()) {
                                try {
                                        Socket socket = serverSocket.accept();
                                        pool.execute(new ClientHandler(socket));

                                } catch (IOException e) {
                                }
                        }
                }
        }

        /**
         * Inner class that handles the servers connection with the clients
         * @author Oskar Engström Magnusson
         *
         */
        private class ClientHandler implements Runnable {
                private Socket socket;
                private ObjectOutputStream oos;
                private ObjectInputStream ois;
                private User user;

                /**
                 * Constructs a ClientHandler and adds a OutputStream and an
InputStream
                 * @param socket the ServerSocket
                 */
                public ClientHandler(Socket socket) {
```

```
                try {
                        this.socket = socket;
                        oos = new
ObjectOutputStream(socket.getOutputStream());
                        ois = new ObjectInputStream(socket.getInputStream());
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }

        /**
         * Method that receives a UserList and writes it to the OutputStream
         * @param users list of users
         */
        public void send(UserList users) {
                try {
                        System.out.println(users);
                        oos.writeObject(users);
                        oos.reset();
                        oos.flush();
                } catch (IOException e) {
                }
                if (unsentMessages.contains(user.getName())) {
                        ArrayList<Message> recievedList =
unsentMessages.get(user.getName());
                        for (Message m : recievedList) {
                                send(m);
                        }
                        unsentMessages.remove(user.getName());
                }
        }

        /**
         * Method that receives a message and writes it to the OutputStream
         * @param message a message object
         */
        public void send(Message message) {
                try {
                        oos.writeObject(message);
                        oos.flush();
                } catch (IOException e) {
                }
        }
```

```
		/**
		 * Method for when a user is denied to login, writes a null object to the
OutputStream
		 * and closes the socket
		 */
		public void denial() {
			try {
				oos.writeObject(null);
				oos.flush();
				socket.close();
			} catch (IOException e) {
				try {
					socket.close();
				} catch (IOException e1) {
					e1.printStackTrace();
				}
			}
		}

		/**
		 * Method that reads an object from the InputStream and chooses what to
do depending
		 * on which object it is
		 */
		public void run() {
			while (!Thread.interrupted() && !socket.isClosed()) {
				try {
					Object request = ois.readObject();
					if (request instanceof User) {
						User user = (User) request;
						if
(clientHandlers.containsKey(user.getName())) {
							denial();
						} else {
							this.user = user;
							this.user.setConnected(true);
							ui.println(this.user.getName() + " has
connected (" + toString() + ")");

clientHandlers.put(this.user.getName(), this);
							// pool.execute(this);
							listener.receive(this.user);
```

```
                                        }
                                } else if (request instanceof Message) {
                                        Message message = (Message) request;
                                        // pool.execute(this);
                                        listener.receive(message);
                                }
                        } catch (SocketException e) {
                                ui.println(user + " has disconnected (" + toString() +
")");
                                clientHandlers.remove(user);
                                listener.disconnectedUser(user);
                                try {
                                        socket.close();
                                } catch (IOException e1) {
                                        // TODO Auto-generated catch block
                                        e1.printStackTrace();
                                }
                        } catch (Exception e) {
                                try {
                                        ui.println(user + " has disconnected (" +
toString() + ")");
                                        socket.close();
                                        clientHandlers.remove(user);
                                        listener.disconnectedUser(user);
                                } catch (IOException e1) {
                                        e1.printStackTrace();
                                }
                        }
                }
        }

        public String toString() {
                return socket.getInetAddress().getHostAddress() + ":" +
socket.getPort();
        }
    }
}
```

ClientListener

```
package ChatServer;

import resources.*;
/**
 * Interface that allows the server to listen to a client
 * @author Oskar Engström Magnusson
 *
 */
public interface ClientListener {

/**
 * Method for receiving a message from a client
 * @param message a message object
 */
public void receive(Message message);

/**
 * Method for receiving a user from a client
 * @param user a User object
 */
public void receive(User user);

/**
 * Method for when a user disconnects from the server
 * @param user a User object
 */
public void disconnectedUser(User user);
}
```

LogUI

```
package ChatServer;

import java.awt.BorderLayout;
import java.awt.Dimension;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/**
 * Class that handles output from Logfile
 * @author Mikael Lindfors
 *
 */
public class LogUI extends JPanel{
	private static final long serialVersionUID = -1598533036400634233L;
	private JTextArea textArea = new JTextArea();
	private JScrollPane jScrollPane = new JScrollPane(textArea);

	/**
	 * Constructor that adds an JScrollPane to the panel and sets the size.
	 */
	public LogUI() {
		this.setLayout(new BorderLayout());
		this.setPreferredSize(new Dimension(400,600));
		this.add(jScrollPane);
	}

	/**
	 * Method that adds strings line by line to the JTextArea
	 * @param logText String one line from the logfile.
	 */
	public void append(String logText) {
		textArea.append(logText + "\n");
	}

	/**
	 * Method that clears the JTextArea.
	 */
	public void clear() {
		textArea.setText("");
```

```
	}
}
```

RunOnThreadN

```
package ChatServer;

import java.util.ArrayList;
import java.util.LinkedList;

/**
 * Class works a general pool of thread that can handle any runnable task.
 */
/**
 *
 * @author Eric Grevillius
 *
 */
public class RunOnThreadN {
	private Buffer<Runnable> tasks = new Buffer<Runnable>();
	private ArrayList<Thread> threads;
	private int n;

	/**
	 * Constructor sets the incoming integer its own variable 'n' which is a
	 * representation of the highest amount of threads.
	 *
	 * @param nbrOfThreads
	 *            Integer representing the number of threads to create.
	 */
	public RunOnThreadN(int nbrOfThreads) {
		this.n = nbrOfThreads;
	}

	/**
	 * Method starts all the threads in the pool only if they aren't started.
	 */
	public synchronized void start() {
		Thread thread;
		if (threads == null) {
			threads = new ArrayList<Thread>();
			for (int i = 0; i < n; i++) {
```

```
                                thread = new Thread() {
                                        public void run() {
                                                Runnable runnable;
                                                while (!Thread.interrupted()) {
                                                        try {
                                                                runnable = tasks.get();
                                                                runnable.run();
                                                                execute(runnable);
                                                        } catch (InterruptedException e) {
                                                                try {
                                                                        join();
                                                                } catch (InterruptedException
e1) {

                                                                }
                                                        }
                                                }
                                        }
                                };
                                thread.start();
                                threads.add(thread);
                        }
                }
        }

        /**
         * Stops all threads in the pool and then clears the pool of all threads.
         */
        public synchronized void stop() {
                if (threads != null) {
                        for (int i = 0; i < n; i++) {
                                execute(new StopThread());
                        }
                        threads.clear();
                        threads = null;
                }
        }

        /**
         * Adds a task in the buffer for the tread-pool to execute
         *
         * @param task
         *            Runnable to but added in the buffer.
         */
```

```
public synchronized void execute(Runnable task) {
	tasks.put(task);
}

/**
 * A Runnable that is put in the buffer when to stop the thread-pool.
 */
/**
 *
 * @author Eric Grevillius
 *
 */
private class StopThread implements Runnable {
	public void run() {
		Thread.currentThread().interrupt();
	}

	public String toString() {
		return "Closing down " + Thread.currentThread();
	}
}

/**
 *
 * @author Eric Grevillius
 *
 *
 *	Class is a buffer for class <T>.
 * @param <T>
 *	  Class to be specified when used.
 */
private class Buffer<T> {
	private LinkedList<T> buffer = new LinkedList<T>();

	/**
	 * Adds given object of the given type to the the buffer.
	 *
	 * @param obj
	 *	    Object of the given type.
	 */
	public synchronized void put(T obj) {
		buffer.addLast(obj);
		notifyAll();
```

```
		}

		/**
		 * Returns an object of the given type.
		 *
		 * @return Object of given type.
		 * @throws InterruptedException
		 */
		public synchronized T get() throws InterruptedException {
			while (buffer.isEmpty()) {
				wait();
			}
			return buffer.removeFirst();
		}
	}
}
```

ServerController

```
package ChatServer;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import resources.*;

/**
 * Class for controlling the server
 * @author Oskar Engström Magnusson
 *
 */
public class ServerController {
	private ChatServer chatServer;
	private ServerUI ui;

	/**
	 * Constructor that creates the server UI and a chatServer-object
	 * @param requestPort the port the server runs through
	 * @param nbrOfThreads number of threads in the thread-pool
	 */
	public ServerController(int requestPort, int nbrOfThreads) {
		ui = new ServerUI(this);
		showServerUI();
```

```
                chatServer = new ChatServer(requestPort, nbrOfThreads, new
ClientResponse(), ui);
        }

        /**
         * Method for showing the server-UI
         */
        private void showServerUI(){
                SwingUtilities.invokeLater(new Runnable(){
                        public void run() {
                                JFrame frame = new JFrame("Server");

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                frame.setVisible(true);
                                frame.add(ui);
                                frame.pack();
                        }
                });
        }

        protected void startServer(){
                chatServer.startServer();
        }

        protected void stopServer(){
                chatServer.stopServer();
        }

        /**
         * Inner class that listens to clients and implements ClientListener
         * @author Oskar Engström Magnusson
         *
         */
        private class ClientResponse implements ClientListener{

                /**
                 * Method that receives a message from a client
                 */
                public void receive(Message message) {
                        ui.println(message);
                        chatServer.respond(message);
                }
```

```
		/**
		 * Method that receives a user from a client
		 */
		public void receive(User user) {
			UserList allUsers = chatServer.getAllUsers();
			if (!allUsers.exist(user.getName())){
				allUsers.addUser(user);
			}
			chatServer.respond(allUsers);
		}

		/**
		 * Method for when a user has disconnected
		 */
		public void disconnectedUser(User user) {
			user.setConnected(false);
		}
	}
}
```

ServerUI

```
package ChatServer;

import javax.swing.*;

import resources.LogWriter;
import resources.LogReader;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 *
 * @author Eric Grevillius
 *
 *	Class that inherits from JPanel and is used as a GUI for seeing all
 *	traffic in the program.
 */
public class ServerUI extends JPanel {
	private static final long serialVersionUID = -2041693529144462758L;
	private ServerController controller;
```

```
private static JTextArea textArea;
private static JScrollPane sp;
private JButton btnStart;
private JButton btnStop;
private JButton btnLog;
private LogWriter log;
private LogUI logUI;
private JFrame logFrame;
private JButton btnClose;
private final String fileName = "files/serverLog.txt";

/**
 * Constructor creates a GUI for seeing the traffic in the server, starting
 * and stopping.
 *
 * @param controller
 *            A link to the controller class.
 */
public ServerUI(ServerController controller) {
        this.controller = controller;
        log = LogWriter.getInstance();
        log.setFileName(fileName);
        logUI = new LogUI();
        int width = 400;
        int height = width;
        Dimension windowSize = new Dimension(width, height);
        this.setPreferredSize(windowSize);
        this.setLayout(new BorderLayout());

        textArea = new JTextArea();
        textArea.setEditable(false);
        sp = new JScrollPane(textArea);
        btnClose = new JButton("Close");
        this.add(sp, BorderLayout.CENTER);
        this.add(buttonPanel(), BorderLayout.SOUTH);
}

/**
 * Creates a panel for the buttons.
 *
 * @return the button-panel
 */
private JPanel buttonPanel() {
```

```
        JPanel panel = new JPanel(new GridLayout(1, 3));
        btnStart = new JButton("Start");
        btnStop = new JButton("Stop");
        btnLog = new JButton("Check log");
        AL aListener = new AL();
        btnStart.addActionListener(aListener);
        btnStop.addActionListener(aListener);
        btnLog.addActionListener(aListener);
        btnClose.addActionListener(aListener);
        panel.add(btnStart, BorderLayout.SOUTH);
        panel.add(btnStop, BorderLayout.SOUTH);
        panel.add(btnLog, BorderLayout.SOUTH);
        return panel;
}

/**
 * Sets the final String of text to the text area
 *
 * @param txt
 *          the final String of text.
 */
public static void setText(final String txt) {
        SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                        textArea.setText(txt);
                        JScrollBar bar = sp.getVerticalScrollBar();
                        bar.setValue(bar.getMaximum() - bar.getVisibleAmount());
                }
        });
}

/**
 * Sets the text of the given object to the text area.
 *
 * @param obj
 *          the given object.
 */
public void setText(Object obj) {
        setText(obj.toString());
}

/**
 * Gives the user options of choosing two time points where the LogUI will
```

```
         * show logged traffic from.
         */
        public void checkLog() {
                String startTime = JOptionPane.showInputDialog("Enter start time
(year:month:date:hour:minute)");
                String endTime = JOptionPane.showInputDialog("Enter end time
(year:month:date:hour:minute)");
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                logFrame = new JFrame("LogUI");
                                JPanel btnPanel = new JPanel();
                                btnPanel.add(btnClose);

logFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                                logFrame.setVisible(true);
                                logFrame.add(logUI);
                                logFrame.pack();
                                logFrame.add(btnPanel, BorderLayout.SOUTH);
                        }
                });
                LogReader logReader = new LogReader(fileName, logUI);
                String logTime = (startTime + ":"+ endTime);
                logReader.read(logTime);
        }

        /**
         * Adds the final String of text to the text area
         *
         * @param txt
         *            the final String of text.
         */
        public void append(final String txt) {
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                textArea.append(txt);
                                log.write(txt);
                        }
                });
        }

        /**
         * Adds the text of the given object to the text area.
         *
```

```
     * @param obj
     *            the given object.
     */
    public void append(Object obj) {
            append(obj.toString());
    }

    /**
     * Adds a empty line in the text area.
     */
    public void println() {
            append("\n");
    }

    /**
     * Adds the given String to the text area and then goes to a new line.
     * @param txt The given String.
     */
    public void println(String txt) {
            append(txt + "\n");
    }

    /**
     * Adds the given objects toString-method to the text area.
     * @param obj
     */
    public void println(Object obj) {
            println(obj.toString());
    }

    /**
     * Inner class that implements an actionlistener.
     *
     * @author Eric Grevillius
     *
     */
    private class AL implements ActionListener {
            public void actionPerformed(ActionEvent e) {
                    if (e.getSource() == btnStart) {
                            controller.startServer();
                    }
                    if (e.getSource() == btnStop) {
                            controller.stopServer();
```

```
                }
                if (e.getSource() == btnLog) {
                        checkLog();
                }
                if (e.getSource() == btnClose) {
                        logUI.clear();
                        logFrame.dispose();
                }
        }
    }

}
```

StartServer

```
package ChatServer;

import javax.swing.SwingUtilities;
/**
 * Starts up the Servers thread and sets up the ip and port for the server to connect to
 * @author Matthias Falk
 *
 */
public class StartServer {
        public static void main(String[] args) {
                SwingUtilities.invokeLater(new Runnable() {
                        public void run() {
                                try {
                                        new ServerController(3300, 10);
                                } catch (Exception e) {
                                        System.out.println("Program: " + e);
                                }
                        }
                });
        }
}
```

UnsentMessages

```
package ChatServer;

import java.util.ArrayList;
import java.util.HashMap;

import resources.Message;
/**
 * Class that saves messages that hasn't been sent in an HashMap. The HashMap
contains an key which is the user name
 * and an ArrayList which holds the message
 * @author Matthias Falk
 *
 */
public class UnsentMessages {
	private HashMap<String, ArrayList<Message>> unsent;

	/**
	 * Declares the unsent variable
	 */
	public UnsentMessages() {
		unsent = new HashMap<String, ArrayList<Message>>();
	}
	/**
	 * Synchronized method that puts an new ArrayList in the HashMap with the
users name as an key
	 * @param user The Users name
	 * @param newList the list that will added
	 */
	public synchronized void put(String user, ArrayList<Message> newList){
		unsent.put(user, newList);
	}
	/**
	 * Synchronized method that returns the ArrayList assigned to the Users name
	 * @param user The Users name
	 * @return an ArrayList which holds the Messages
	 */
	public synchronized ArrayList<Message> get(String user){
		return unsent.get(user);
	}
	/**
```

```
	 * Synchronized method that removes all Messages assigned to the Users name
	 * @param user The name of the User
	 */
	public synchronized void remove (String user) {
		unsent.remove(user);
	}
	/**
	 * Synchronized method that returns true if the HashMap contains the given key
	 * @param key The key that will be checked
	 * @return boolean
	 */
	public synchronized boolean contains(String key){
		return unsent.containsKey(key);
	}
}
```

Källkod klient

ChatClient

```
package ChatClient;
import java.io.*;
import java.net.*;
import javax.swing.JOptionPane;
import resources.*;
/**
 * Class that opens a Socket to our Server and starts the ServerListener that
 * will recieve Messages and a User list.
 *
 * @author Martin Gyllstrom
 *
 */
public class ChatClient {
 private String ip;
 private int serverPort;
 private Socket socket;
 private ObjectOutputStream oos;
 private ObjectInputStream ois;
 private ServerListener listener;
 /**
  * Constructor that opens to the server and starts an ObjectInputStream to the
  * socket and a serverlistener.
  *
  * @param ip
  *        String with the ip-address to the selected server
  * @param serverPort
  *        Port number the server is listening at.
```

```
 * @param listener
 *        that adds a listener to the MessageList.
 */
protected ChatClient(String ip, int serverPort, ServerListener listener) {
 this.ip = ip;
 this.serverPort = serverPort;
 this.listener = listener;
}
protected void login(User user) {
 try {
  this.socket = new Socket(ip, serverPort);
  oos = new ObjectOutputStream(socket.getOutputStream());
  oos.flush();
  ois = new ObjectInputStream(socket.getInputStream());
  oos.writeObject(user);
  oos.flush();
  new TCPListener().start();
 } catch (IOException e) {
  e.printStackTrace();
  JOptionPane.showMessageDialog(null, "Can not connect to server \n" + e.getMessage());
 }
}
/**
 * Method that sends a message to the chosen port.
 *
 * @param message
 *        sending message to selected server.
 */
protected void send(Message message) {
 try {
```

```
  oos.writeObject(message);
  oos.flush();
 } catch (IOException e) {
  JOptionPane.showMessageDialog(null, "Message could not be sent\n" + e.getMessage());
 }
}
/**
 * Inner class that creates a thread so the client recieves an Object from the
 * server. When an Object is recieved it will be sent to the UserList.
 *
 * @author Martin Gyllstrom
 *
 */
private class TCPListener extends Thread {
 public void run() {
  while (!socket.isClosed()) {
   try {
    Object response = ois.readObject();
    System.out.println(response);
    if (response instanceof UserList) {
     UserList list = (UserList) response;
     listener.receive(list);
    } else if (response instanceof Message) {
     Message message = (Message) response;
     listener.receive(message);
    } else if (response == null) {
     listener.accessDenied();
    }
   } catch (ClassNotFoundException | IOException e) {
    e.printStackTrace();
```

```
    JOptionPane.showMessageDialog(null, "TCPLISTENER interrupted" + e.getMessage());
    try {
     socket.close();
    } catch (IOException e1) {
    }
   }
  }
 }
}
```

ClientController

```
package ChatClient;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;
import java.util.ArrayList;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
import resources.Message;
import resources.User;
import resources.UserList;
/**
 * Class that creates a client for the program and controls it.
 *
 * @author Martin Gyllstrom
 *
 */
public class ClientController {
 private LoginUI loginUI;
 private MessageUI messageUI;
 private ChatClient chatClient;
 private JFrame UI;
 private User user;
 private UserList allUsers;
 private UserList contacts;
```

```
private String generateFileName() {
 String fileName;
 fileName = "files/localContacts" + user + ".dat";
 return fileName;
}
/**
 * Constructor that opens to the server
 *
 * @param ip String with the ip-address to the selected server
 *
 * @param ServerPort Port number the server is listening at.
 */
protected ClientController(String ip, int serverPort) {
 chatClient = new ChatClient(ip, serverPort, new ServerResponse());
 contacts = new UserList();
 allUsers = new UserList();
 loginUI = new LoginUI(this);

 showLoginUI();
}
/**
 * Method that show the LogIn screen.
 */

private void showLoginUI() {
 SwingUtilities.invokeLater(new Runnable() {
  public void run() {
   UI = new JFrame("Login");
   UI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   UI.setVisible(true);
   UI.add(loginUI);
   UI.pack();
  }
 });
}
/**
 * Method that show the Message screen.
 */

private void showMessageUI() {
 SwingUtilities.invokeLater(new Runnable() {
  public void run() {
   UI = new JFrame("Messenger (" + user + ")");
```

```
    UI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    UI.setVisible(true);
    UI.add(messageUI);
    UI.pack();
    UI.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent arg0) {
      writeContacts();
     }
    });
   }
  });
 }
 /**
  * Method that reads the UserList from the file.
  */

 public void readContacts() {
  String fileName = generateFileName();
  try (ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new
FileInputStream(fileName)))) {
   try {
    contacts = (UserList) ois.readObject();
   } catch (ClassNotFoundException e) {
   }
  } catch (IOException e) {
  }
 }
 /**
  * Method that writes the contacts to a file.
  */

 private void writeContacts() {
  String fileName = generateFileName();
  try (ObjectOutputStream oos = new ObjectOutputStream(
   new BufferedOutputStream(new FileOutputStream(fileName)))) {
   oos.writeObject(contacts);
   oos.flush();
  } catch (IOException e) {
   e.printStackTrace();
  }
 }
 protected User getThisUser() {
  return user;
```

```
}
protected UserList getContacts() {
 return contacts;
}
protected UserList getAllUsers() {
 return allUsers;
}
protected void login() {
 String name = loginUI.getName();
 String iconPath = loginUI.getIconPath();
 ImageIcon icon;
 if (iconPath.equals("")) {
  this.user = new User(name);
 } else {
  icon = new ImageIcon(loginUI.getIconPath());
  this.user = new User(name, icon);
 }
 chatClient.login(user);
}
/**
 * Method that sends the message
 */

protected void send() {
 String text = messageUI.getText();
 String iconPath = messageUI.getImagePath();
 ImageIcon image;
 if (iconPath.equals("")) {
  image = null;
 } else {
  image = new ImageIcon(messageUI.getImagePath());
 }
 UserList receivers = new UserList();
 ArrayList<String> usernames = messageUI.getReceivers();
 for (int i = 0; i < usernames.size(); i++) {
  if (allUsers.exist(usernames.get(i))) {
   int index = allUsers.indexOf(usernames.get(i));
   receivers.addUser(allUsers.getUser(index));
  } else if(contacts.exist(usernames.get(i))){
   int index = contacts.indexOf(usernames.get(i));
   receivers.addUser(contacts.getUser(index));
  }
 }
```

```
 Message message = new Message(user, receivers, text, image);
 chatClient.send(message);
}
void addContact(String contact) {
 if (contacts.exist(contact)) {
  JOptionPane.showMessageDialog(null, contact + " already added");
 } else {
  if (allUsers.exist(contact)) {
   contacts.addUser(allUsers.getUser(allUsers.indexOf(contact)));
   allUsers.removeUser(allUsers.getUser(allUsers.indexOf(contact)));
   JOptionPane.showMessageDialog(null, contact + " added");
  } else {
   JOptionPane.showMessageDialog(null, contact + " doesn't exists");
  }
 }
 messageUI.populateContactList();
 messageUI.populateOnlineList();
 messageUI.revalidate();
 messageUI.repaint();
}
/**
 * Inner class that implements a serverlistener that waits to recieve an
 * message.
 *
 * @author Martin Gyllstrom
 *
 */

private class ServerResponse implements ServerListener {
 public void receive(Message message) {
  messageUI.addResponse(message);
 }
 public void receive(UserList userList) {
  allUsers = userList;
  if (allUsers.exist(user.getName())) {
   allUsers.removeUser(user);
  }
  if (user.isConnected()) {
   messageUI.populateOnlineList();
   messageUI.revalidate();
   messageUI.repaint();
  } else {
   user.setConnected(true);
```

```
    messageUI = new MessageUI(ClientController.this);
    UI.dispose();
    showMessageUI();
   }
  }
  public void accessDenied() {
   JOptionPane.showMessageDialog(null,
     "Access denied! \n \n User already online." + "\n (wait a minute and try again)");
  }
 }
}
```

LoginUI

```
package ChatClient;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
/**
 * Class that extends the JPanel and creates an LogIn-screen for the program.
 */
/**
 *
 * @author Eric Grevillius
 *
 */
public class LoginUI extends JPanel{
 private static final long serialVersionUID = -6025240258441519294L;
 private ClientController controller;
 private JPanel pnlContent = new JPanel(new GridLayout(2,1,0,0));
 private JPanel pnlButtons = new JPanel(new GridLayout(1,2,5,0));
 private JButton btnImage = new JButton("Choose Icon");
 private JButton btnLogin = new JButton("Log In");
 private JTextField tfName = new JTextField();
 private JPanel pnlTextField;
 private JPanel pnlBtnImage;
 private JPanel pnlBtnLogin;
 private JFileChooser fc;
 private File file;
```

```
private JLabel lblFile = new JLabel();

/**
 * Creates an Log in-screen
 *
 * @param cont a link to the controller
 */

public LoginUI(ClientController cont) {
 this.controller = cont;
 pnlBtnLogin = new JPanel();
 pnlBtnImage = new JPanel();
 pnlTextField = new JPanel();
 int width = 250;
 int height = 100;
 Dimension windowSize = new Dimension(width,height);
 this.setPreferredSize(windowSize);
 this.setLayout(new BorderLayout());
 fc = new JFileChooser(new File(System.getProperty("user.home"), "Pictures"));
 fc.setDialogTitle("Image chooser");
 FileFilter imageFilter = new FileNameExtensionFilter(
         "Image files", ImageIO.getReaderFileSuffixes());
 fc.addChoosableFileFilter(imageFilter);
 fc.setAcceptAllFileFilterUsed(false);

 pnlContent.add(pnlTextField);
 tfName.setToolTipText("Name");
 tfName.setColumns(20);
 pnlTextField.add(tfName);
 pnlButtons.add(pnlBtnImage);
 btnImage.setToolTipText("Image");
 pnlBtnImage.add(btnImage);

 pnlButtons.add(pnlBtnLogin);
 btnLogin.setToolTipText("Login");
 pnlBtnLogin.add(btnLogin);

 pnlContent.add(pnlButtons);
 this.add(pnlContent, BorderLayout.CENTER);
 lblFile.setPreferredSize(new Dimension(0, 20));
 this.add(lblFile, BorderLayout.SOUTH);

 initializeListeners();
```

```
}

private void initializeListeners(){
 btnImage.addActionListener(new AL());
 btnLogin.addActionListener(new AL());
}

public String getName(){
 String name = tfName.getText();
 return name;
}

/**
 * Method that receives a file
 * @return the filename
 */

public String getIconPath(){
 String filename = lblFile.getText();
 return filename;
}

/**
 * Inner class that implements an actionlistener.
 *
 * @author
 *
 */

private class AL implements ActionListener{
 public void actionPerformed(ActionEvent e) {
  if (e.getSource() == btnImage){
   int fileValue = fc.showSaveDialog(null);
   if (fileValue == JFileChooser.APPROVE_OPTION){
    file = fc.getSelectedFile();
    lblFile.setText(file.getAbsolutePath());
   }
  }
  if (e.getSource() == btnLogin){
   controller.login();
  }
 }
```

```
 }
}
```

MessageUI

```
package ChatClient;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.*;
import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import resources.*;
public class MessageUI extends JPanel {
 private static final long serialVersionUID = 6091921199167131315L;
 private ClientController controller;
 private JPanel pnlRead;
 private JPanel pnlWrite;
 private JPanel pnlUsers;
 private JTextArea taRead;
 private JTextArea taWrite;
 private JButton btnSend;
 private JButton btnImage;
 private JButton btnAddContact;
 private JLabel lblImageFile;
 private JScrollPane scrollReadPane;
 private JScrollPane scrollContactPane;
 private JFileChooser fc;
 private File file;
 private JPanel pnlContacts;
 private JPanel pnlOnline;
 private JList<String> listContacts;
 private JList<String> listOnline;
 private JPanel pnlContainer;
 private ArrayList<String> receivers = new ArrayList<String>();
 private JPanel pnlProfile;
 private JLabel lblIcon;
 private DefaultListModel<String> modelOnlineList = new DefaultListModel<>();
 private DefaultListModel<String> modelContactList = new DefaultListModel<>();
 public MessageUI(ClientController cont) {
```

```
 this.controller = cont;
 Dimension windowSize = new Dimension(500, 500);
 this.setPreferredSize(windowSize);
 this.setLayout(new BorderLayout());
 pnlWrite = writePanel();
 pnlRead = readPanel();
 pnlUsers = contactPanel();
 pnlUsers.setPreferredSize(new Dimension(150, 0));
 this.add(pnlRead, BorderLayout.CENTER);
 this.add(pnlWrite, BorderLayout.SOUTH);
 this.add(pnlUsers, BorderLayout.EAST);
 Boolean editable = true;
 taRead.setEditable(!editable);
 taWrite.setEditable(editable);
 initializeListeners();
 controller.readContacts();
 populateContactList();
}
private JPanel writePanel() {
 taWrite = new JTextArea();
 btnSend = new JButton("Send");
 JPanel panel = new JPanel();
 panel.setLayout(new BorderLayout());
 panel.setPreferredSize(new Dimension(10, 100));
 panel.add(writeImagePanel(), BorderLayout.NORTH);
 panel.add(taWrite, BorderLayout.CENTER);
 panel.add(btnSend, BorderLayout.EAST);
 return panel;
}
private JPanel readPanel() {
 taRead = new JTextArea();
 scrollReadPane = new JScrollPane(taRead);
 JPanel panel = new JPanel();
 panel.setLayout(new BorderLayout());
 panel.add(scrollReadPane);
 return panel;
}
private JPanel contactPanel() {
 JPanel panel = new JPanel();
 panel.setLayout(new BorderLayout(0, 0));
 pnlContainer = new JPanel();
 scrollContactPane = new JScrollPane(pnlContainer);
 panel.add(scrollContactPane);
```

```
  pnlContainer.setLayout(new GridLayout(0, 1, 0, 0));
  pnlContacts = new JPanel();
  pnlContainer.add(pnlContacts);
  pnlContacts.setLayout(new BorderLayout(0, 0));
  JLabel lblContacts = new JLabel("Contacts");
  lblContacts.setHorizontalAlignment(SwingConstants.CENTER);
  pnlContacts.add(lblContacts, BorderLayout.NORTH);
  listContacts = new JList<String>(modelContactList);
  populateContactList();
  listContacts.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
  pnlContacts.add(listContacts, BorderLayout.CENTER);
  pnlOnline = new JPanel();
  pnlContainer.add(pnlOnline);
  pnlOnline.setLayout(new BorderLayout(0, 0));
  JLabel lblOnline = new JLabel("Online");
  pnlOnline.add(lblOnline, BorderLayout.NORTH);
  lblOnline.setHorizontalAlignment(SwingConstants.CENTER);
  // 2 NYA RADER
  listOnline = new JList<>(modelOnlineList);
  populateOnlineList();
  listOnline.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
  pnlOnline.add(listOnline, BorderLayout.CENTER);
  pnlProfile = new JPanel();
  panel.add(pnlProfile, BorderLayout.NORTH);
  pnlProfile.setLayout(new BorderLayout(0, 0));
  btnAddContact = new JButton("+ Add Contact");
  pnlProfile.add(btnAddContact, BorderLayout.SOUTH);
  lblIcon = new JLabel(new
ImageIcon(controller.getThisUser().getPicture().getImage().getScaledInstance(150, 150,
Image.SCALE_SMOOTH)));
  lblIcon.setHorizontalAlignment(SwingConstants.CENTER);
  lblIcon.setToolTipText(controller.getThisUser().getName());
  lblIcon.setMaximumSize(new Dimension(100, 100));
  pnlProfile.add(lblIcon, BorderLayout.CENTER);
  return panel;
 }
 private JPanel writeImagePanel() {
  lblImageFile = new JLabel("");
  btnImage = new JButton("Choose image");
  fc = new JFileChooser(new File(System.getProperty("user.home"), "Pictures"));
  fc.setDialogTitle("Image chooser");
  FileFilter imageFilter = new FileNameExtensionFilter("Image files",
ImageIO.getReaderFileSuffixes());
```

```
 fc.addChoosableFileFilter(imageFilter);
 fc.setAcceptAllFileFilterUsed(false);
 JPanel panel = new JPanel(new BorderLayout());
 panel.add(btnImage, BorderLayout.WEST);
 panel.add(lblImageFile, BorderLayout.CENTER);
 return panel;
}
private void initializeListeners() {
 Listener l = new Listener();
 btnSend.addActionListener(l);
 btnImage.addActionListener(l);
 btnAddContact.addActionListener(l);
}
protected void populateContactList() {
 UserList list = controller.getContacts();
 for (int i = 0; i < list.numberOfUsers(); i++) {
  if (!modelContactList.contains(list.getUser(i).getName())) {
   modelContactList.addElement(list.getUser(i).getName());
  }
 }
}
protected void populateOnlineList() {
 UserList list = controller.getAllUsers();
 for (int i = 0; i < list.numberOfUsers(); i++) {
  if (!modelOnlineList.contains(list.getUser(i).getName())) {
   modelOnlineList.addElement(list.getUser(i).getName());
  }
 }
}
protected String getText() {
 return taWrite.getText();
}
protected String getImagePath() {
 return lblImageFile.getText();
}
protected ArrayList<String> getReceivers() {
 return receivers;
}
public void addResponse(Message message) {
 String content = taRead.getText();
 String append = "From " + message.getSender().getName() + " : " + message.getText();
 this.taRead.setText(content + append + "\n");
 ImageIcon image = message.getImage();
```

```
  if (image != null) {
   JOptionPane.showMessageDialog(null, null, "Message from " +
message.getSender().getName(),
     JOptionPane.PLAIN_MESSAGE, image);
  }
 }
 private class Listener implements ActionListener, KeyListener {
  @Override
  public void keyPressed(KeyEvent k) {
  }
  @Override
  public void keyReleased(KeyEvent k) {
  }
  @Override
  public void keyTyped(KeyEvent k) {
  }
  @Override
  public void actionPerformed(ActionEvent a) {
   if (a.getSource() == btnImage) {
    int fileValue = fc.showSaveDialog(null);
    if (fileValue == JFileChooser.APPROVE_OPTION) {
     file = fc.getSelectedFile();
     lblImageFile.setText(file.getPath());
    }
   }
   if (a.getSource() == btnSend) {
    java.util.List<String> listC = listContacts.getSelectedValuesList();
    java.util.List<String> listO = listOnline.getSelectedValuesList();
    for (int i = 0; i < listO.size() || i < listC.size(); i++) {
     if (i < listO.size()) {
      if (!receivers.contains(listO.get(i))){
       receivers.add(listO.get(i));
      }
     }
     if (i < listC.size()) {
      if (!receivers.contains(listC.get(i))){
       receivers.add(listC.get(i));
      }
     }
    }
    controller.send();
    receivers.clear();
    lblImageFile.setText("");
```

```
    taWrite.setText("");
   }
   if (a.getSource() == btnAddContact) {
    controller.addContact(JOptionPane.showInputDialog("Search for User"));
   }
  }
 }
}
```

ServerListener

```
package ChatClient;
import resources.*;
public interface ServerListener {
public void receive(Message message);
public void receive(UserList userList);
public void accessDenied();
}
```

StartClient

```
package ChatClient;
import javax.swing.SwingUtilities;
public class StartClient {
 public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
   public void run() {
    try {
     new ClientController("localhost",3300);
    } catch(Exception e) {
     System.out.println("Program: "+e);
    }
   }
  });
 }
}
```

Källkod för gemensamma klasser i klient och server

LogReader

```
package resources;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.Calendar;

import ChatServer.LogUI;

/**
 * Class that handles reading from Logfile
 *
 * @author Mikael Lindfors
 *
 */
public class LogReader {
	private String filename;
	private LogUI logUI;

	/**
	 * Constructor that receives filename and LogUI object
	 *
	 * @param filename String filename of the logfile.
	 * @param logUI LogUI object for displaying text in the LogUI's JTextArea
	 */
	public LogReader(String filename, LogUI logUI) {
```

```
		this.filename = filename;
		this.logUI = logUI;

	}

	/**
	 * Method that takes a String in format
	 *
fromYear:fromMonth:fromDay:fromHour:fromMinute:tillYear:tillMonth:tillDay:tillHour:tillMinute
	 * and splits it to seperate parts and adds it to Calendarobjects that will be
	 * compared to a timestamp (also added to a Calendarobject) from the logfile. If
	 * the timestamp from Logfile is between timeFrom and timeTill, log-output will
	 * be send to the LogUI's JTextArea.
	 *
	 * @param logDate
	 */
	public synchronized void read(String logDate) {
		Calendar timeFrom = Calendar.getInstance();
		Calendar timeTill = Calendar.getInstance();
		Calendar timeInLog = Calendar.getInstance();

		try (BufferedReader reader = new BufferedReader(
				new InputStreamReader(new FileInputStream(filename),
"UTF-8"))) {
			int[] dateUI = logStringToInt(logDate.split(":"));
			timeFrom.set(dateUI[0], dateUI[1], dateUI[2], dateUI[3], dateUI[4]);
			timeTill.set(dateUI[5], dateUI[6], dateUI[7], dateUI[8], dateUI[9]);
			String line = reader.readLine();
			while (line != null) {
				int[] timeFromLog = logStringToInt(parseFromLog(line));
				timeInLog.set(timeFromLog[0], timeFromLog[1], timeFromLog[2],
timeFromLog[3], timeFromLog[4]);
				if (timeFrom.before(timeInLog) && timeTill.after(timeInLog)) {
```

```
                                logUI.append(line.toString());
                        }
                        line = reader.readLine();
                }
        } catch (Exception e) {
                System.err.println(e);
        }
}

/**
 * Method that fetch the timestamp from a String line (from the logfile).
 *
 * @param line String with format (year-month-day hour:minute:second)
 * @return return a String array with year,month,day,hour,minute,second as
 *         separate parts.
 */
private String[] parseFromLog(String line) {
        String[] parts = new String[5];
        parts[0] = line.substring(1, 5);
        parts[1] = line.substring(6, 8);
        parts[2] = line.substring(9, 11);
        parts[3] = line.substring(12, 14);
        parts[4] = line.substring(15, 17);
        return parts;
}

/**
 * Method that takes a String array with both fromTime and tillTime and converts
 * it to an int-array.
 *
 * @param logDate String array with format [0] = fromYear, [1] = fromMonth, [2]
 *           = fromDay, [3] = fromHour, [4] = fromMinute, [5] = tillYear, [6] =
 *           tillMonth, [7] = tillDay, [8] = tillHour, [9] = tillMinute.
```

```
	 *
	 */
	public int[] logStringToInt(String[] logDate) {
		int[] parts = new int[logDate.length];
		for (int index = 0; index < logDate.length; index++) {
			parts[index] = Integer.parseInt(logDate[index]);
		}
		return parts;    }
}
```

LogWriter

```
package resources;

import java.io.*;
import java.util.Calendar;
/**
 * Class that handles log-output and write it to the harddrive.
 * @author Mikael Lindfors
 *
 */

public class LogWriter {
	private static LogWriter instance;
	private String filename;
	private Calendar c;
	private int year, month, day, hour, minute, second;

	/**
	 * Constructor that fetch an instance from Calendar.
	 */
	private LogWriter() {
		c = Calendar.getInstance();
	}

	/**
	 * Method that returns an instance of LogWriter (singleton)
	 * @return LogWriter instance.
	 */
	public static LogWriter getInstance() {
		if (instance == null) {
			instance = new LogWriter();
		}
		return instance;
	}

	/**
	 * Method that sets the fileName for the logfile.
	 * @param filename String with the filename.
	 */
	public void setFileName(String filename) {
		this.filename = filename;
```

```
	}

	/**
	 * Method that fetch the time from the Calendar object and format it in a correct way
	 * @return String with the current time in format: year-month-day hour:minute:second
	 *
	 */
	private String getTime() {
		year = c.get(Calendar.YEAR);
		month = c.get(Calendar.MONTH) + 1;
		day = c.get(Calendar.DATE);
		hour = c.get(Calendar.HOUR_OF_DAY);
		minute = c.get(Calendar.MINUTE);
		second = c.get(Calendar.SECOND);
		String time = year + "-" + (month < 10 ? ("0" + month) : (month)) + "-" + (day < 10
? ("0" + day) : (day)) + " "
					+ (hour < 10 ? ("0" + hour) : (hour)) + ":" + (minute < 10 ? ("0" +
minute) : (minute)) + ":"
					+ (second < 10 ? ("0" + second) : (second));
		return time;
	}
/**
 * Method that writes a String line to the logfile together with a correct timestamp.
 * @param textToFile String with one line of text.
 */
	public synchronized void write(String textToFile) {
		try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new
FileOutputStream(filename, true)))) {
			writer.write("(" + getTime() + ") " + textToFile);
			writer.flush();
		} catch (Exception e) {
			System.err.println(e);
		}
	}
}
```

Message

```
package resources;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import javax.swing.ImageIcon;

/**
 * A class that represents a message
 *
 * @author elinolsson
 *
 */
public class Message implements Serializable {
	private static final long serialVersionUID = 1516088037980799826L;
	private User sender;
	private UserList receivers;
	private String text;
	private ImageIcon image;
	private String delivered;
	private String received;

	/**
	 * Constructor
	 *
	 * @param sender
	 *            - the sender of the message
	 * @param receivers
	 *            - the recievers of the message
	 */
	public Message(User sender, UserList receivers) {
		this.sender = sender;
		this.receivers = receivers;
	}

	/**
	 *
	 * @param sender
	 *            -the sender of the message
	 * @param receivers
```

```
	 *		- the recievers of the message
	 * @param text
	 *		- the text sent
	 */
	public Message(User sender, UserList receivers, String text) {
		this(sender, receivers);
		this.text = text;
	}

	/**
	 *
	 * @param sender
	 *		-the sender of the message
	 * @param receivers
	 *		- the recievers of the message
	 * @param text
	 *		- the text sent in a message
	 * @param image - the image sent in a message
	 */
	public Message(User sender, UserList receivers, String text, ImageIcon image) {
		this(sender, receivers, text);
		this.image = image;
	}
/**
 * Getter for the sender
 * @return sender - the sender
 */
	public User getSender() {
		return sender;
	}
/**
 * Getter for the recievers
 * @return recievers - the recevers
 */
	public UserList getReceivers() {
		return receivers;
	}
/**
 * Setter for the text sent
 * @param text - the text sent
 */
	public void setText(String text) {
		this.text = text;
```

```
	}
/**
 * Getter for the text sent
 * @param text - the text sent
 */
	public String getText() {
		return text;
	}
/**
 * Setter for the image sent
 * @param image - the image sent
 */
	public void setImage(ImageIcon image) {
		this.image = image;
	}
	/**
	 * Getter for the image sent
	 * @param image - the image sent
	 */
	public ImageIcon getImage() {
		return image;
	}
/**
 * Sets the date that the message was delivered
 */
	public void setDeliveredDate() {
		if (delivered == null) {
			this.delivered =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
		}
	}
/**
 * Getter for the date that the message was delivered
 * @return delivered - the date
 */
	public String getDeliveredDate() {
		return delivered;
	}
	/**
	 * Setter for the date that the message was recieved
	 * @return recieved - the date
	 */
	public void setReceivedDate() {
```

```
                if (received == null) {
                        this.received =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"));
                }
        }
/**
 *  Getter for the date that the message was recieved
 * @return recieved - the date
 */
        public String getReceivedDate() {
                return received;
        }
/**
 * toString method that prints a message
 * @return a string
 */
        public String toString() {
                return "(" + super.toString() + ") Sender: " + sender + " Receivers: " + receivers +
" Text: " + text
                                + " Image: " + image + " Delivered: " + delivered + " Received: " +
received;
        }
}
```

User

```
package resources;

import java.io.Serializable;

import javax.swing.*;
/**
 * A Class represents a User
 * @author elinolsson
 *
 */
public class User implements Serializable{
	private static final long serialVersionUID = -773659708777609854L;
	private String name;
	private ImageIcon picture;
	private boolean connected;

	/**
	 * Constructor
	 * @param name - the Users name
	 * Creates a new picture for the user
	 */
	public User(String name){
		this.name = name;
		this.picture = new ImageIcon("images/no_icon.png");
	}

	/**
	 * Constructor
	 * @param name - the users name
	 * @param pic - the users profile picture
	 * if the user dont choose a pic, a standardicon picture will be given.
	 */
	public User(String name, ImageIcon pic){
		this.name = name;
		this.picture = pic;
	}
/**
 * Getter for the users name
 * @return - the users name
 */
```

```
	public String getName(){
		return name;
	}
	/**
	 * Getter for the user picture
	 * @return - the users picture
	 */
	public ImageIcon getPicture(){
		return picture;
	}
	/**
	 * Method that return the hashCode
	 * @return - hashcode
	 */
	public int hashCode(){
		return name.hashCode();
	}
	/**
	 * Method that returns true or false if users name equals the incomming Object anObject
	 * @return true or false
	 */
	public boolean equals(Object anObject){
		return name.equals(anObject);
	}
	/**
	 * Setter for the users profile picture
	 * @param pic - the users picture
	 */
	public void setPicture(ImageIcon pic){
		this.picture = pic;
	}
	/**
	 * Boolean, returns true or false if the user is connected
	 * @return connected - either true or false
	 */
	public boolean isConnected() {
		return connected;
	}
/**
 * Setter for the current value of connected, either true or false
 * @param connected - the result of isConnected
 */
	public void setConnected(boolean connected) {
```

```
		this.connected = connected;
	}
	/**
	 * Method that returns the users name
	 * @return name - the users name
	 */
	public String toString(){
		return name;
	}
}
```

UserList

```
package resources;

/**
 * A class that represents a userlist. All methods is synchonized which is a security matter
 * @author elinolsson
 */
import java.io.Serializable;
import java.util.ArrayList;

public class UserList implements Serializable {
	private static final long serialVersionUID = -1968868156826365885L;
	private ArrayList<User> users;

	/**
	 * constructor creates a arraylist of users
	 */
	public UserList() {
		users = new ArrayList<User>();
	}

	/**
	 * returns the number of users in the list
	 *
	 * @return - the size of the array
	 */
	public synchronized int numberOfUsers() {
		return users.size();
	}
```

```
/**
 * adds a user to the list
 *
 * @param user
 *            - the user to be added
 */
public synchronized void addUser(User user) {
        users.add(user);
}

/**
 * removes a user form the list
 *
 * @param user
 *            - the user to be removed
 */
public synchronized void removeUser(User user) {
        int index = indexOf(user.getName());
        users.remove(index);
}

/**
 * Getter for the userlist
 *
 * @return - the list
 */
public synchronized UserList getUserList() {
        return this;
}

/**
 * gets the user at the i position
 *
 * @param i
 *            - the position in the array
 * @return - the user at pos i
 */
public synchronized User getUser(int i) {
        return users.get(i);
}

/**
 * if the user exist in the list returns true, otherwise false
```

```
 *
 * @param username
 *            - the user name
 * @return true or false
 */
public synchronized boolean exist(String username) {
	for (User u : users) {
		if (username.equals(u.getName())) {
			return true;
		}
	}
	return false;
}

/**
 * returns the index of the user. if the user doesn't exist in the list, -1
 * will be returned
 *
 * @param username
 *            - the username
 * @return either i or -1
 */
public synchronized int indexOf(String username) {
	for (int i = 0; i < users.size(); i++) {
		if (username.equals(users.get(i).getName())) {
			return i;
		}
	}
	return -1;
}

/**
 * Method that prints out the users in the list
 *
 * @return the string
 */
public synchronized String toString() {
	String str = "";
	for (int i = 0; i < users.size(); i++) {
		str += users.get(i);
		if (i < users.size() - 1) {
			str += ", ";
		}
```

```
                }
                return str;

        }
}
```