JAVA CHAT MESSENGER

TABLE OF CONTENTS

ABSTRACT

Communication is a mean for people to exchange messages. Teleconferencing or Chatting, is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years but the acceptance it was quite recent. Our project is an example of a primitive chat messenger. It is made up of two important parts i.e., the client and the server. There can be around 100 servers used simultaneously and 50 clients can join a group chat at a particular time. One of the important features is the clients can easily switch from one server to other easily by changing their server port numbers. This multi-client chat application is mainly composed of sockets and threads.

INTRODUCTION

Communication is a mean for people to exchange messages. It has started since the beginning of human creation. Distant communication began as early as 1800 century with the introduction of television, telegraph and then telephony. Interestingly enough, telephone communication stands out as the fastest growing technology, from fixed line to mobile wireless, from voice call to data transfer. The emergence of computer network and telecommunication technologies bear the same objective that is to allow people to communicate. All this while, much efforts has been drawn towards consolidating the device into one and therefore indiscriminate the services. Chatting is a method of using technology to bring people and ideas together despite of the geographical barriers. The technology has been available for years but the acceptance it was quite recent. Our project is an example of a chat server. It is made up of applications the client application which runs on the users mobile and server application which runs on any pc on the network. To start chatting our client should get connected to server where they can do Group and private chatting.

Let's us see a brief history of online communication before we hop into the contents of our project:

- MIT creates first computer-based messaging system (May 1962)
- Marty Cooper makes the first phone call (April 1972)
- Community memory in Berkley California becomes first computer bulletin board system (August 1972)
- Community bulletin board system, Chicago (Feb 1978)
- Weight of popularity (1980's to mid-1990's)
- World wide web (1990)
- SMS Invented (Dec 1982)
- ICO Messenger (November 1996)
- AOL Launched AUM (May 1997)
- Yahoo messenger released (March 1998)
- MSN messenger released (July 1999)
- iChat launches (Aug 2002)
- Skype launches (Aug 2002)
- Black cherry messenger launched (2005)
- Google talk launched (Aug 2006)
- iPhone launched (June 2007)
- WhatsApp (October 2008)
- Facebook launches messenger (Aug 2010)
- Snapchat hits the scene (May 2012)
- Smartphone sales surprises (Dec 2012)
- Voice recognition (Feb 2013)
- Facebook acquires WhatsApp (April 2014)

THE PROBLEM STATEMENT

This project aims to design a simple chat messenger. The socket networking and multithreading are major concepts used. This project tries to accommodate the maximum number of OOP concepts being implemented.

THE PROPOSED APPROACH

The proposed approach of chat messenger used the multi- threading mechanism to handle multiple servers and multiple clients at a time. The code is designed in such a way that it can handle about 300 servers simultaneously and each server is designed in such a way that it can handle 50 clients simultaneously.

The following are the in-built modules used in this approach:

- **java.awt:** awt package is the main package of the AWT, or Abstract Windowing Toolkit. It contains classes for graphics, including the Java 2D graphics capabilities introduced in the Java 2 platform, and also defines the basic graphical user interface (GUI) framework for Java.
- **java.swing:** Java Swing tutorial is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike, AWT, Java Swing provides platform-independent and lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.
- **java.io:** The Java I/O package, a.k.a. java.io, provides a set of input streams and a set of output streams used to read and write data to files or other input and output sources. There are three categories of classes in java.io: input streams, output streams and everything else.
- **java.text:** The java. text package consists of classes and interfaces that are useful for writing internationalized programs that handle local customs, such as date and time formatting and string alphabetization, correctly. This package is new as of Java 1.1.
- **java.util:** It contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array). Following are the Important Classes in Java.
- **java.net:** The package java.net contains classes and interfaces that provide a powerful infrastructure for networking in Java. These include: The URL class for basic access to Uniform Resource Locators (URLs). The URLConnection class, which supports more complex operations on URLs.

CODE:

There are totally 5 classes created namely:

- ServerUI
- ClientUI
- TextAreaOutputStream
- Freepostfinder
- ChatServerCLI
- ChatClientCLI

ServerUI class:

```java
package UI;

import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import me.alexpanov.net.FreePortFinder;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;

import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import java.awt.Color;

public class ServerUI extends JFrame implements ActionListener {

	// Socket Related
	public static SimpleDateFormat formatter = new SimpleDateFormat("[MM/hh/yy
hh:mm a]");
  //public static SimpleDateFormat formatter = new SimpleDateFormat("[hh:mm a]");
	private static HashMap<String, PrintWriter> connectedClients = new
HashMap<>();
	private static final int MAX_CONNECTED = 50;
	private static int PORT;
	private static ServerSocket server;
	private static volatile boolean exit = false;

	// JFrame related
	private JPanel contentPane;
	private JTextArea txtAreaLogs;
	private JButton btnStart;
	private JLabel lblChatServer;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					ServerUI frame = new ServerUI();

	UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFee
l");
					SwingUtilities.updateComponentTreeUI(frame);
					//Logs
					System.setOut(new PrintStream(new
TextAreaOutputStream(frame.txtAreaLogs)));
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the frame.
	 */
	public ServerUI() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 570, 400);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(new BorderLayout(0, 0));

		lblChatServer = new JLabel("CHAT SERVER");
		lblChatServer.setHorizontalAlignment(SwingConstants.CENTER);
		lblChatServer.setFont(new Font("Tahoma", Font.PLAIN, 40));
		contentPane.add(lblChatServer, BorderLayout.NORTH);

		btnStart = new JButton("START");
		btnStart.addActionListener(this);
		btnStart.setFont(new Font("Tahoma", Font.PLAIN, 30));
		contentPane.add(btnStart, BorderLayout.SOUTH);

		JScrollPane scrollPane = new JScrollPane();
		contentPane.add(scrollPane, BorderLayout.CENTER);

		txtAreaLogs = new JTextArea();
		txtAreaLogs.setBackground(Color.CYAN);
		txtAreaLogs.setForeground(Color.BLACK);
```

```java
		txtAreaLogs.setLineWrap(true);
		scrollPane.setViewportView(txtAreaLogs);
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if(e.getSource() == btnStart) {
			if(btnStart.getText().equals("START")) {
				exit = false;
				getRandomPort();
				start();
				btnStart.setText("STOP");
			}else {
				addToLogs("Chat server stopped...");
				exit = true;
				btnStart.setText("START");
			}
		}

		//Refresh UI
		refreshUIComponents();
	}

	public void refreshUIComponents() {
		lblChatServer.setText("CHAT SERVER" + (!exit ? ": "+PORT:""));
	}

	public static void start() {
		new Thread(new ServerHandler()).start();
	}

	public static void stop() throws IOException {
		if (!server.isClosed()) server.close();
	}

	private static void broadcastMessage(String message) {
		for (PrintWriter p: connectedClients.values()) {
			p.println(message);
		}
	}

	public static void addToLogs(String message) {
		System.out.printf("%s %s\n", formatter.format(new Date()), message);
	}

	private static int getRandomPort() {
		int port = FreePortFinder.findFreeLocalPort();
		PORT = port;
		return port;
	}

	private static class ServerHandler implements Runnable{
		@Override
		public void run() {
			try {
				server = new ServerSocket(PORT);
				addToLogs("Server started on port: " + PORT);
				addToLogs("Now listening for connections...");
				while(!exit) {
```

```
                    if (connectedClients.size() <= MAX_CONNECTED){
                        new Thread(new
ClientHandler(server.accept())).start();
                    }
                }
            }
            catch (Exception e) {
                addToLogs("\nError occured: \n");
                addToLogs(Arrays.toString(e.getStackTrace()));
                addToLogs("\nExiting...");
            }
        }
    }

    // Start of Client Handler
    private static class ClientHandler implements Runnable {
        private Socket socket;
        private PrintWriter out;
        private BufferedReader in;
        private String name;

        public ClientHandler(Socket socket) {
            this.socket = socket;
        }

        @Override
        public void run(){
            addToLogs("Client connected: " + socket.getInetAddress());
            try {
                in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                out = new PrintWriter(socket.getOutputStream(), true);
                for(;;) {
                    name = in.readLine();
                    if (name == null) {
                        return;
                    }
                    synchronized (connectedClients) {
                        if (!name.isEmpty() &&
!connectedClients.keySet().contains(name)) break;
                        else out.println("INVALIDNAME");
                    }
                }
                out.println("Welcome to the chat group, " +
name.toUpperCase() + "!");
                addToLogs(name.toUpperCase() + " has joined.");
                broadcastMessage("[SYSTEM] " + name.toUpperCase() + "
has joined.");
                connectedClients.put(name, out);
                String message;
                out.println("You may join the chat now...");
                while ((message = in.readLine()) != null && !exit) {
                    if (!message.isEmpty()) {
                        if (message.toLowerCase().equals("/quit"))
break;

                        broadcastMessage(String.format("[%s] %s",
name, message));
                    }
                }
                } catch (Exception e) {
                        addToLogs(e.getMessage());
                } finally {
                        if (name != null) {
                                addToLogs(name + " is leaving");
                                connectedClients.remove(name);
                                broadcastMessage(name + " has left");
                        }
                }
        }
    }

}
```

ClientUI class:

```
package UI;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Date;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Font;
import javax.swing.JButton;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import java.awt.Color;
import java.awt.FlowLayout;
import javax.swing.JTextField;

public class ClientUI extends JFrame implements ActionListener {

        // Socket Related
        private static Socket clientSocket;
        private static int PORT;
        private PrintWriter out;

        // JFrame related
        private JPanel contentPane;
        private JTextArea txtAreaLogs;
        private JButton btnStart;
        private JPanel panelNorth;
        private JLabel lblChatClient;
        private JPanel panelNorthSouth;
```

```
	private JLabel lblPort;
	private JLabel lblName;
	private JPanel panelSouth;
	private JButton btnSend;
	private JTextField txtMessage;
	private JTextField txtNickname;
	private JTextField txtPort;
	private String clientName;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					ClientUI frame = new ClientUI();

	UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFee
l");
					SwingUtilities.updateComponentTreeUI(frame);
					//Logs
					System.setOut(new PrintStream(new
TextAreaOutputStream(frame.txtAreaLogs)));
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Create the frame.
	 */
	public ClientUI() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 570, 400);
		contentPane = new JPanel();
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);
		contentPane.setLayout(new BorderLayout(0, 0));

		panelNorth = new JPanel();
		contentPane.add(panelNorth, BorderLayout.NORTH);
		panelNorth.setLayout(new BorderLayout(0, 0));

		lblChatClient = new JLabel("CHAT CLIENT");
		lblChatClient.setHorizontalAlignment(SwingConstants.CENTER);
		lblChatClient.setFont(new Font("Tahoma", Font.PLAIN, 40));
		panelNorth.add(lblChatClient, BorderLayout.NORTH);

		panelNorthSouth = new JPanel();
		panelNorth.add(panelNorthSouth, BorderLayout.SOUTH);
		panelNorthSouth.setLayout(new FlowLayout(FlowLayout.RIGHT, 5, 5));

		lblName = new JLabel("Nickname");
		panelNorthSouth.add(lblName);
		txtNickname = new JTextField();
		txtNickname.setColumns(10);
		panelNorthSouth.add(txtNickname);

		lblPort = new JLabel("Port");
		panelNorthSouth.add(lblPort);

		txtPort = new JTextField();
		panelNorthSouth.add(txtPort);
		txtPort.setColumns(10);

		btnStart = new JButton("START");
		panelNorthSouth.add(btnStart);
		btnStart.addActionListener(this);
		btnStart.setFont(new Font("Tahoma", Font.PLAIN, 12));

		JScrollPane scrollPane = new JScrollPane();
		contentPane.add(scrollPane, BorderLayout.CENTER);

		txtAreaLogs = new JTextArea();
		txtAreaLogs.setBackground(Color.PINK);
		txtAreaLogs.setForeground(Color.BLACK);
		txtAreaLogs.setLineWrap(true);
		scrollPane.setViewportView(txtAreaLogs);

		panelSouth = new JPanel();
		FlowLayout fl_panelSouth = (FlowLayout) panelSouth.getLayout();
		fl_panelSouth.setAlignment(FlowLayout.RIGHT);
		contentPane.add(panelSouth, BorderLayout.SOUTH);

		txtMessage = new JTextField();
		panelSouth.add(txtMessage);
		txtMessage.setColumns(50);

		btnSend = new JButton("SEND");
		btnSend.addActionListener(this);
		btnSend.setFont(new Font("Tahoma", Font.PLAIN, 12));
		panelSouth.add(btnSend);
	}

	@Override
	public void actionPerformed(ActionEvent e) {
		if(e.getSource() == btnStart) {
			if(btnStart.getText().equals("START")) {
				btnStart.setText("STOP");
				start();
			}else {
				btnStart.setText("START");
				stop();
			}
		}else if(e.getSource() == btnSend) {
			String message = txtMessage.getText().trim();
			if(!message.isEmpty()) {
				out.println(message);
				txtMessage.setText("");
			}
		}
		//Refresh UI
		refreshUIComponents();
	}

	public void refreshUIComponents() {

	}

	public void start() {
		try {
			PORT = Integer.parseInt(txtPort.getText().trim());
			clientName = txtNickname.getText().trim();
			clientSocket = new Socket("localhost", PORT);
			out = new PrintWriter(clientSocket.getOutputStream(), true);
			new Thread(new Listener()).start();
			//send name
			out.println(clientName);
		} catch (Exception err) {
			addToLogs("[ERROR] "+err.getLocalizedMessage());
		}
	}

	public void stop(){
		if(!clientSocket.isClosed()) {
			try {
				clientSocket.close();
			} catch (IOException e1) {}
		}
	}

	public static void addToLogs(String message) {
		System.out.printf("%s %s\n", ServerUI.formatter.format(new Date()), 
message);
	}

	private static class Listener implements Runnable {
		private BufferedReader in;
		@Override
		public void run() {
			try {
				in = new BufferedReader(new 
InputStreamReader(clientSocket.getInputStream()));
				String read;
				for(;;) {
					read = in.readLine();
					if (read != null && !(read.isEmpty())) 
addToLogs(read);
				}
			} catch (IOException e) {
				return;
			}
		}
	}
}
```

TextAreaOutputStream class:

```java
package UI;

import java.io.IOException;
import java.io.OutputStream;
import javax.swing.JTextArea;

public class TextAreaOutputStream extends OutputStream {

	private final JTextArea txtArea;
    private final StringBuilder sb = new StringBuilder();

	public TextAreaOutputStream(JTextArea txtArea) {
		this.txtArea = txtArea;
	}

	@Override
	public void write(int b) throws IOException {
		if (b == '\r') {
	          return;
	  }
	  if (b == '\n') {
	      final String text = sb.toString() + "\n";

	      txtArea.append(text);
	      sb.setLength(0);
	  } else {
	      sb.append((char) b);
	  }
	}
}
```

Freeportfinder class:

```java
package me.alexpanov.net;

import java.io.IOException;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.util.NoSuchElementException;
import java.util.concurrent.atomic.AtomicInteger;

/**
 * Finds currently available server ports.
 */
public final class FreePortFinder {

    /**
     * The minimum server currentMinPort number for IPv4.
     * Set at 1100 to avoid returning privileged currentMinPort numbers.
     */
    public static final int MIN_PORT_NUMBER = 1100;

    /**
     * The maximum server currentMinPort number for IPv4.
     */
    public static final int MAX_PORT_NUMBER = 65535;

    /**
     * We'll hold open the lowest port in this process
     * so parallel processes won't use the same block
     * of ports.   They'll go up to the next block.
     */
    private static final ServerSocket LOCK;

    /**
     * Incremented to the next lowest available port when findFreeLocalPort() is
called.
     */
    private static AtomicInteger currentMinPort = new
AtomicInteger(MIN_PORT_NUMBER);

    /**
     * Creates a new instance.
     */
    private FreePortFinder() {
        // Do nothing
    }

    static {
        int port = MIN_PORT_NUMBER;
        ServerSocket ss = null;

        while (ss == null) {
            try {
                ss = new ServerSocket(port);
            } catch (Exception e) {
                ss = null;
                port += 200;
            }
        }
        LOCK = ss;
        Runtime.getRuntime().addShutdownHook(new Thread() {
            public void run() {
                try {
                    LOCK.close();
                } catch (Exception ex) {
                    //ignore
                }
            }
        });
        currentMinPort.set(port + 1);
    }

    /**
     * Gets the next available port starting at the lowest number. This is the
preferred
     * method to use. The port return is immediately marked in use and doesn't
rely on the caller actually opening
     * the port.
     *
     * @throws IllegalArgumentException is thrown if the port number is out of
range
     * @throws NoSuchElementException if there are no ports available
     * @return the available port
     */
    public static synchronized int findFreeLocalPort() {
        return findFreeLocalPort(null);
    }

    /**
     * Gets the next available port starting at the lowest number. This is the
preferred
     * method to use. The port return is immediately marked in use and doesn't
rely on the caller actually opening
     * the port.
     *
     * @param bindAddress the address that will try to bind
     * @throws IllegalArgumentException is thrown if the port number is out of
range
     * @throws NoSuchElementException if there are no ports available
     * @return the available port
     */
    public static synchronized int findFreeLocalPort(InetAddress bindAddress) {
        int next = findFreeLocalPort(currentMinPort.get(), bindAddress);
        currentMinPort.set(next + 1);
        return next;
    }

    /**
     * Gets the next available port starting at the lowest number. This is the
preferred
     * method to use. The port return is immediately marked in use and doesn't
rely on the caller actually opening
     * the port.
     *
     * @throws IllegalArgumentException is thrown if the port number is out of
range
     * @throws NoSuchElementException if there are no ports available
     * @return the available port
     */
    public static synchronized int findFreeLocalPort(int fromPort) {
        return findFreeLocalPort(fromPort, null);
    }

    /**
     * Gets the next available port starting at a given from port.
     *
     * @param fromPort the from port to scan for availability
     * @param bindAddress the address that will try to bind
     * @throws IllegalArgumentException is thrown if the port number is out of
range
     * @throws NoSuchElementException if there are no ports available
     * @return the available port
     */
    public static synchronized int findFreeLocalPort(int fromPort, InetAddress
bindAddress) {
        if (fromPort < currentMinPort.get() || fromPort > MAX_PORT_NUMBER) {
            throw new IllegalArgumentException("From port number not in valid
range: " + fromPort);
        }

        for (int i = fromPort; i <= MAX_PORT_NUMBER; i++) {
            if (available(i, bindAddress)) {
                return i;
            }
        }

        throw new NoSuchElementException("Could not find an available port above "
+ fromPort);
    }

    /**
     * Gets the next available port starting at a given from port.
     *
     * @param bindAddresses the addresses that will try to bind
     * @throws IllegalArgumentException is thrown if the port number is out of
range
     * @throws NoSuchElementException if there are no ports available
     * @return the available port
     */
    public static synchronized int findFreeLocalPortOnAddresses(InetAddress ...
bindAddresses) {
        int fromPort = currentMinPort.get();
        if (fromPort < currentMinPort.get() || fromPort > MAX_PORT_NUMBER) {
            throw new IllegalArgumentException("From port number not in valid
range: " + fromPort);
        }
        if (bindAddresses != null) {
            for (int j = fromPort; j <= MAX_PORT_NUMBER; j++) {
                for (int i = 0; i < bindAddresses.length ; i++) {
                    if (available(j, bindAddresses[i])) {
                        currentMinPort.set(j + 1);
                        return j;
                    }
                }
            }
        }

        throw new NoSuchElementException("Could not find an available port above "
+ fromPort);
    }
        /**
         * Checks to see if a specific port is available.
         *
         * @param port the port number to check for availability
         * @return <tt>true</tt> if the port is available, or <tt>false</tt> if
not
         * @throws IllegalArgumentException is thrown if the port number is out of
range
         */
    public static boolean available(int port) throws IllegalArgumentException {
        return available(port, null);
    }

    /**
     * Checks to see if a specific port is available.
     *
     * @param port the port number to check for availability
     * @param bindAddress the address that will try to bind
     * @return <tt>true</tt> if the port is available, or <tt>false</tt> if not
     * @throws IllegalArgumentException is thrown if the port number is out of
range
```

```
     */
    public static boolean available(int port, InetAddress bindAddress) throws
IllegalArgumentException {
        if (port < currentMinPort.get() || port > MAX_PORT_NUMBER) {
            throw new IllegalArgumentException("Invalid start currentMinPort: " +
port);
        }

        ServerSocket ss = null;
        DatagramSocket ds = null;
        try {
            ss = (bindAddress != null) ? new ServerSocket(port, 50, bindAddress) :
new ServerSocket(port);
            ss.setReuseAddress(true);
            ds = (bindAddress != null) ? new DatagramSocket(port, bindAddress) :
new DatagramSocket(port);
            ds.setReuseAddress(true);
            return true;
        } catch (IOException e) {
            // Do nothing
        } finally {
            if (ds != null) {
                ds.close();
            }

            if (ss != null) {
                try {
                    ss.close();
                } catch (IOException e) {
                    /* should not be thrown */
                }
            }
        }

        return false;

    }
}
```

ChatServerCLI class:

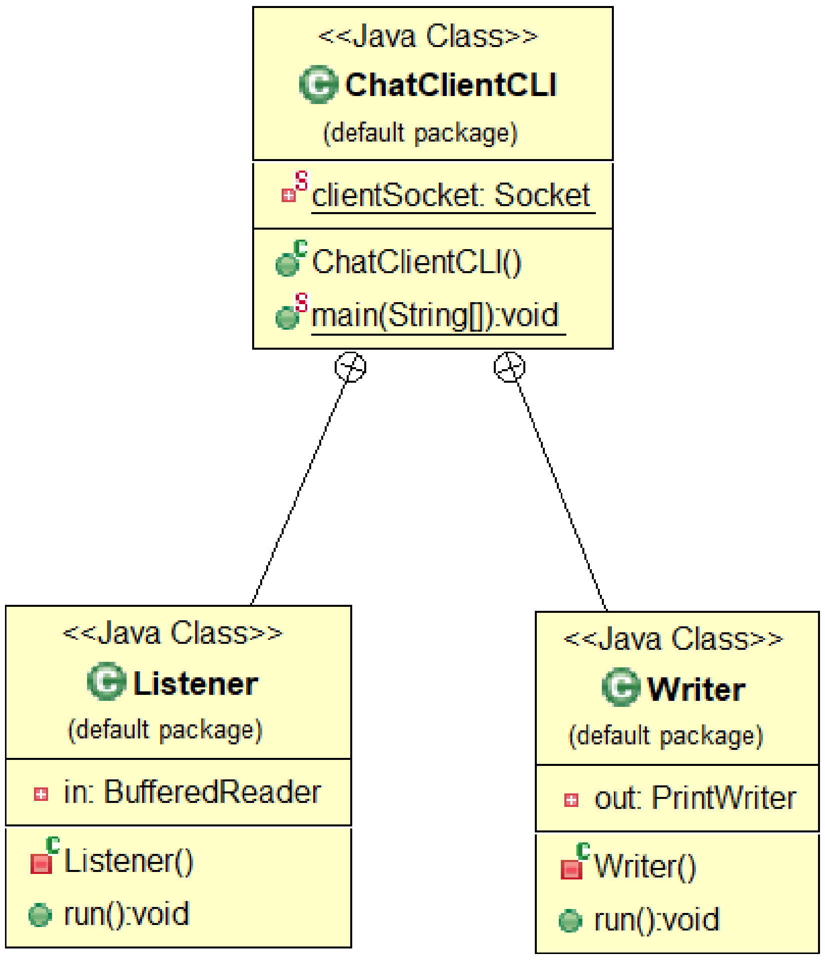
```
import java.io.*;
import java.net.*;
import java.util.HashMap;

import me.alexpanov.net.FreePortFinder;

public class ChatServerCLI {
	private static HashMap<String, PrintWriter> connectedClients = new
HashMap<>();
	private static final int MAX_CONNECTED = 50;
	private static int PORT;
	private static boolean verbose;
	private static ServerSocket server;

	// Start of Client Handler
	private static class ClientHandler implements Runnable {
		private Socket socket;
		private PrintWriter out;
		private BufferedReader in;
		private String name;

		public ClientHandler(Socket socket) {
			this.socket = socket;
		}

		@Override
		public void run(){
			if (verbose)
				System.out.println("Client connected: " +
socket.getInetAddress());
			try {
				in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
				out = new PrintWriter(socket.getOutputStream(), true);
				for(;;) {
					out.println("Enter username:\t");
					name = in.readLine();
					if (name == null) {
						return;
					}
					synchronized (connectedClients) {
						if (!name.isEmpty() &&
!connectedClients.keySet().contains(name)) break;
						else out.println("INVALIDNAME");
					}
				}
				out.println("Welcome to the chat group, " +
name.toUpperCase() + "!");
				if (verbose) System.out.println(name.toUpperCase() + "
has joined.");
				broadcastMessage("[SYSTEM MESSAGE] " +
name.toUpperCase() + " has joined.");
				connectedClients.put(name, out);
				String message;
				out.println("You may join the chat now...");
				while ((message = in.readLine()) != null) {
					if (!message.isEmpty()) {
						if (message.toLowerCase().equals("/quit"))
break;

							broadcastMessage("palaigit si

hamima");

						}
						broadcastMessage(name + ": " + message);
					}
				}
			} catch (Exception e) {
				if (verbose) System.out.println(e);
			} finally {
				if (name != null) {
					if (verbose) System.out.println(name + " is
leaving");

					connectedClients.remove(name);
					broadcastMessage(name + " has left");
				}
			}
		}
```

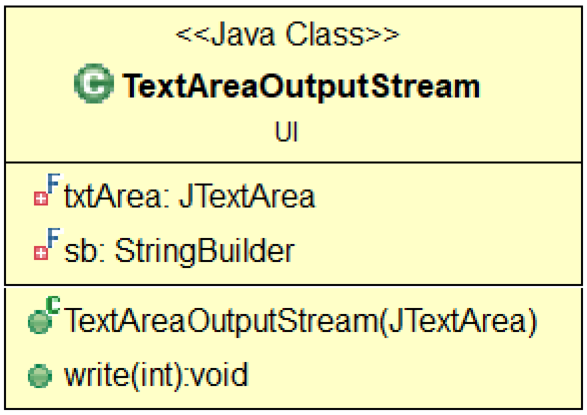
```java
        }
        // End of Client Handler

        private static void broadcastMessage(String message) {
                for (PrintWriter p: connectedClients.values()) {
                        p.println(message);
                }
        }

        public static void start(boolean isVerbose) {
                verbose = isVerbose;
                try {
                        server = new ServerSocket(getRandomPort());
                        if (verbose) {
                                System.out.println("Server started on port: " + PORT);
                                System.out.println("Now listening for connections...");
                        }
                        for(;;) {
                                if (connectedClients.size() <= MAX_CONNECTED){
                                        Thread newClient = new Thread(
                                                        new ClientHandler(server.accept()));
                                        newClient.start();
                                }
                        }
                }
                catch (Exception e) {
                        if (verbose) {
                                System.out.println("\nError occured: \n");
                                e.printStackTrace();
                                System.out.println("\nExiting...");
                        }
                }
        }

        public static void stop() throws IOException {
                if (!server.isClosed()) server.close();
        }

        private static int getRandomPort() {
                int port = FreePortFinder.findFreeLocalPort();
                PORT = port;
                return port;
        }

        public static void main(String[] args) throws IOException {
                start(args[0].toLowerCase().equals("verbose") ? true : false);
        }


}
```

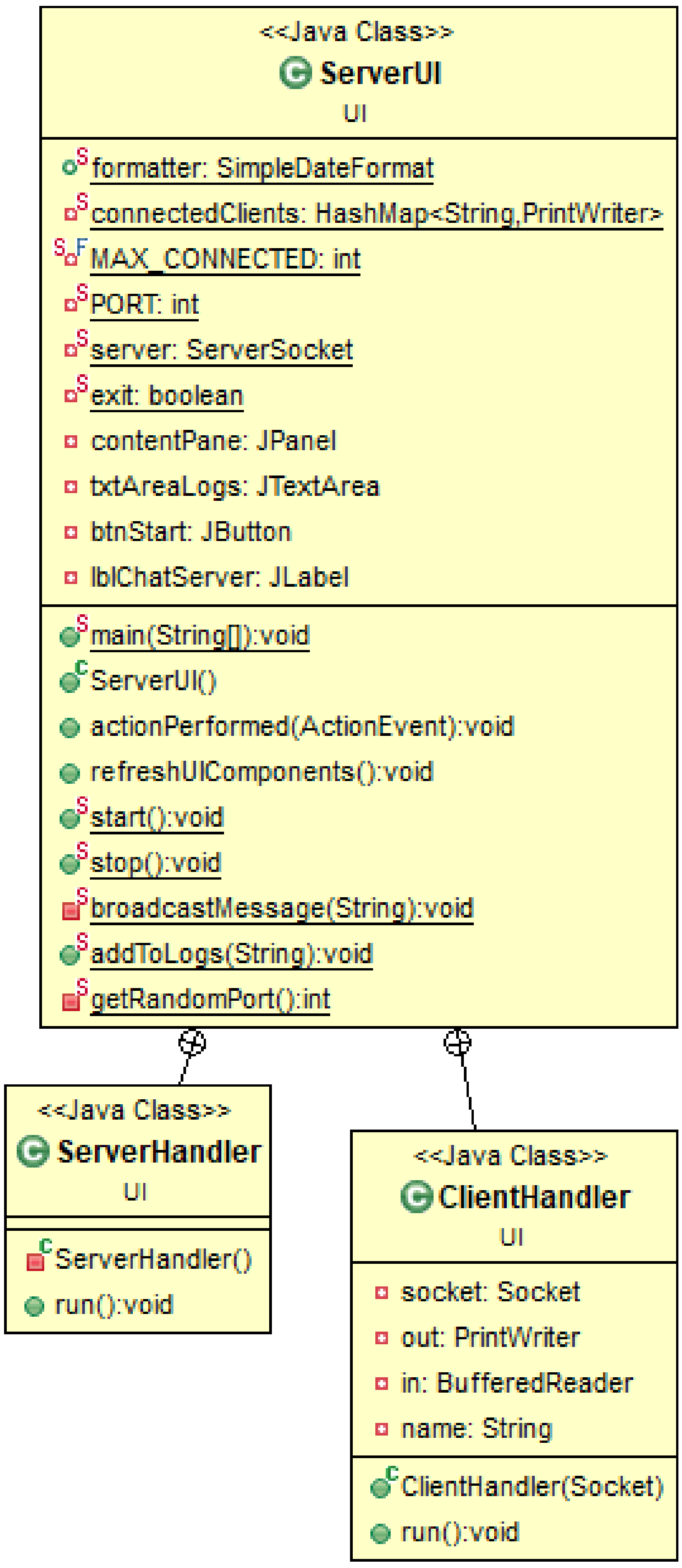
ChatClientCLI class:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class ChatClientCLI {
	private static Socket clientSocket;
	private static class Listener implements Runnable {
		private BufferedReader in;
		@Override
		public void run() {
			try {
				in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
				String read;
				for(;;) {
					read = in.readLine();
					if (read != null && !(read.isEmpty()))
System.out.println(read);
				}
			} catch (IOException e) {
				return;
			}
		}

	}

	// Writer class:
	// implements the run method from Runnable to enable thread creation for
the
	// writing portion of a server to client communication

	private static class Writer implements Runnable {
		private PrintWriter out;
		@Override
		public void run() {
			Scanner write = new Scanner(System.in);
			try {
				out = new PrintWriter(clientSocket.getOutputStream(),
true);
				for(;;) {
					if (write.hasNext())
out.println(write.nextLine());
				}
			} catch (IOException e) {
				write.close();
				return;
			}
		}
	}

	public static void main(String[] args) {
		String ipAddress = null;
		if(args.length != 0) {
			ipAddress = args[0];
		} else {
			ipAddress = "localhost";
		}
		try {
			// this will try to create socket connection if server socket
exists
			clientSocket = new Socket(ipAddress, 4378);
		} catch (Exception e) {
			// throws an error if there is no server socket in that port
			e.printStackTrace();
		}
		new Thread(new Writer()).start();
		new Thread(new Listener()).start();
	}
}
```

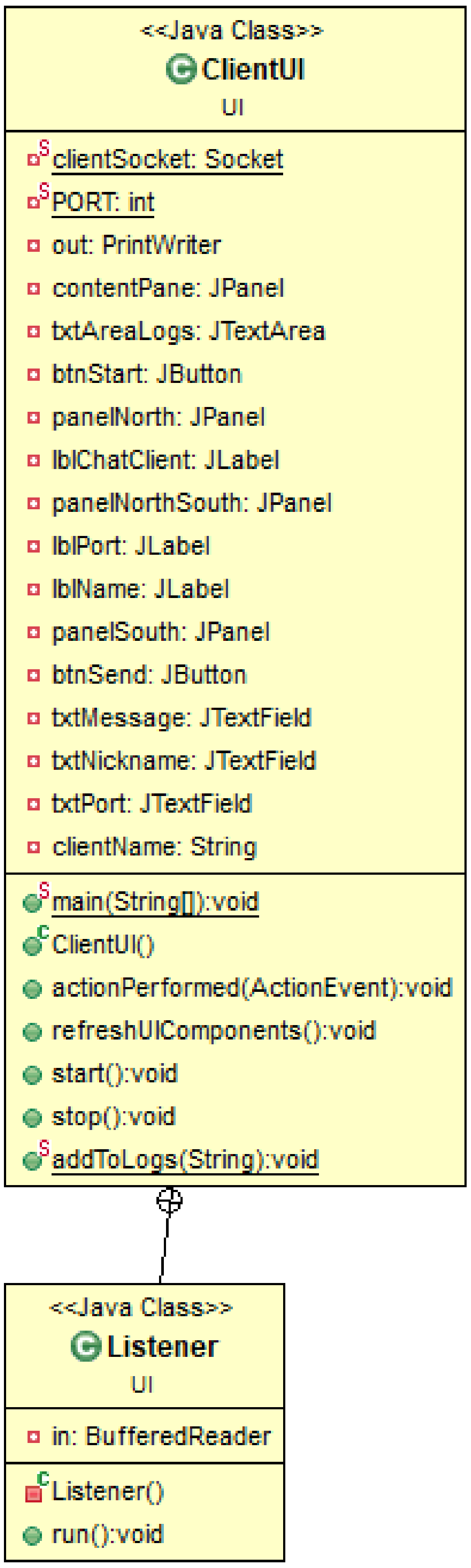
The UML diagram of every class:

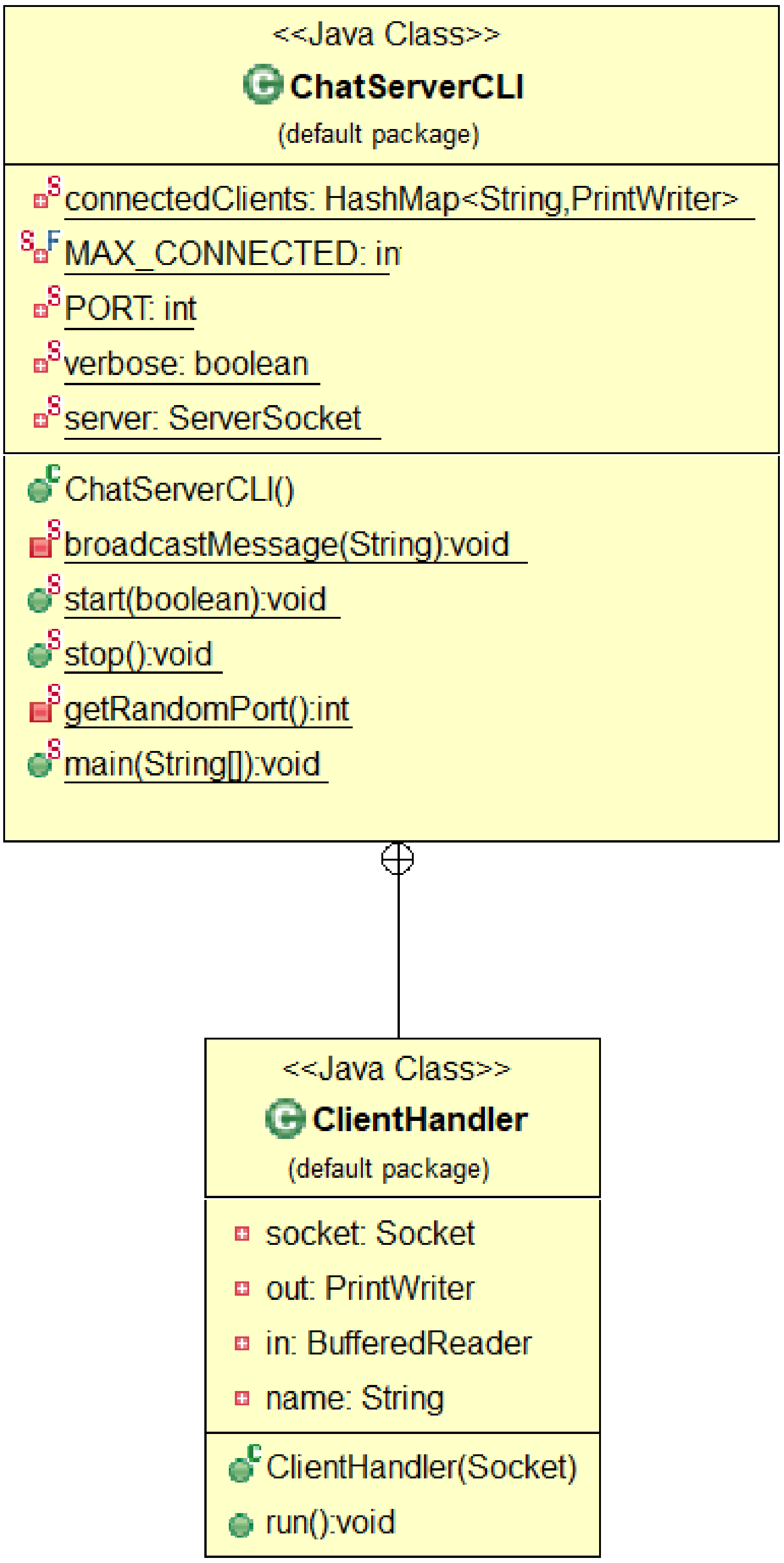




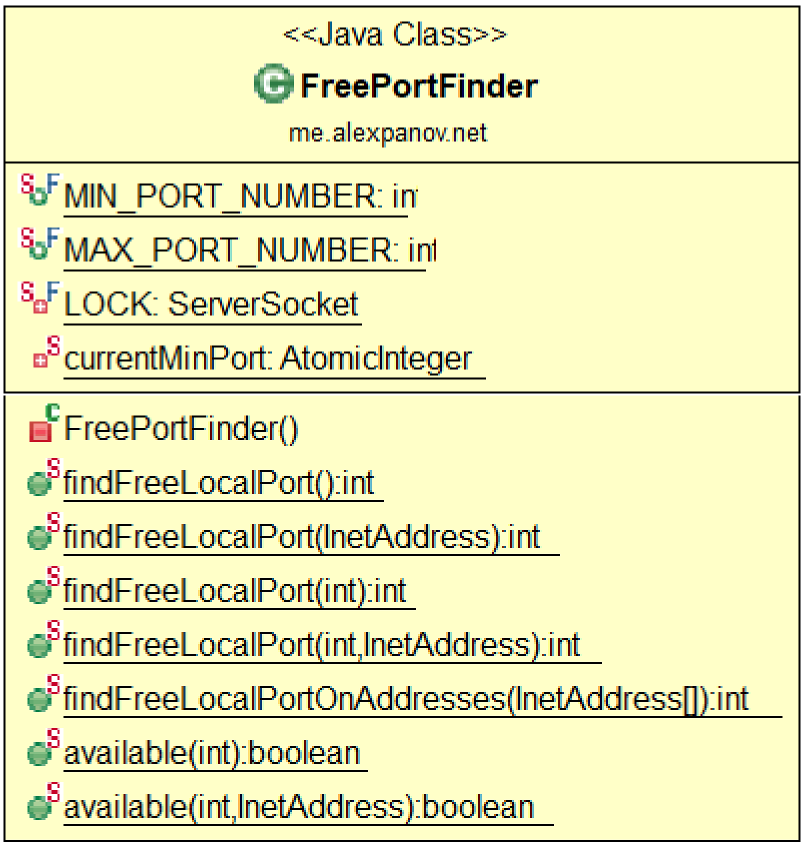


<<Java Class>>
ServerUI
UI
formatter: SimpleDateFormat
connectedClients: HashMap<String,PrintWriter>
MAX_CONNECTED: int
PORT: int
server: ServerSocket
exit: boolean
contentPane: JPanel
txtAreaLogs: JTextArea
btnStart: JButton
lblChatServer: JLabel
main(String[]):void
ServerUI()
actionPerformed(ActionEvent):void
refreshUIComponents():void
start():void
stop():void
broadcastMessage(String):void
addToLogs(String):void
getRandomPort():int
<<Java Class>>
ServerHandler
UI
ServerHandler()
run():void
<<Java Class>>
ClientHandler
UI
socket: Socket
out: PrintWriter
in: BufferedReader
name: String
ClientHandler(Socket)
run():void


<<Java Class>>
ClientUI
UI
clientSocket: Socket
PORT: int
out: PrintWriter
contentPane: JPanel
txtAreaLogs: JTextArea
btnStart: JButton
panelNorth: JPanel
lblChatClient: JLabel
panelNorthSouth: JPanel
lblPort: JLabel
lblName: JLabel
panelSouth: JPanel
btnSend: JButton
txtMessage: JTextField
txtNickname: JTextField
txtPort: JTextField
clientName: String
main(String[]):void
ClientUI()
actionPerformed(ActionEvent):void
refreshUIComponents():void
start():void
stop():void
addToLogs(String):void
<<Java Class>>
Listener
UI
in: BufferedReader
Listener()
run():void

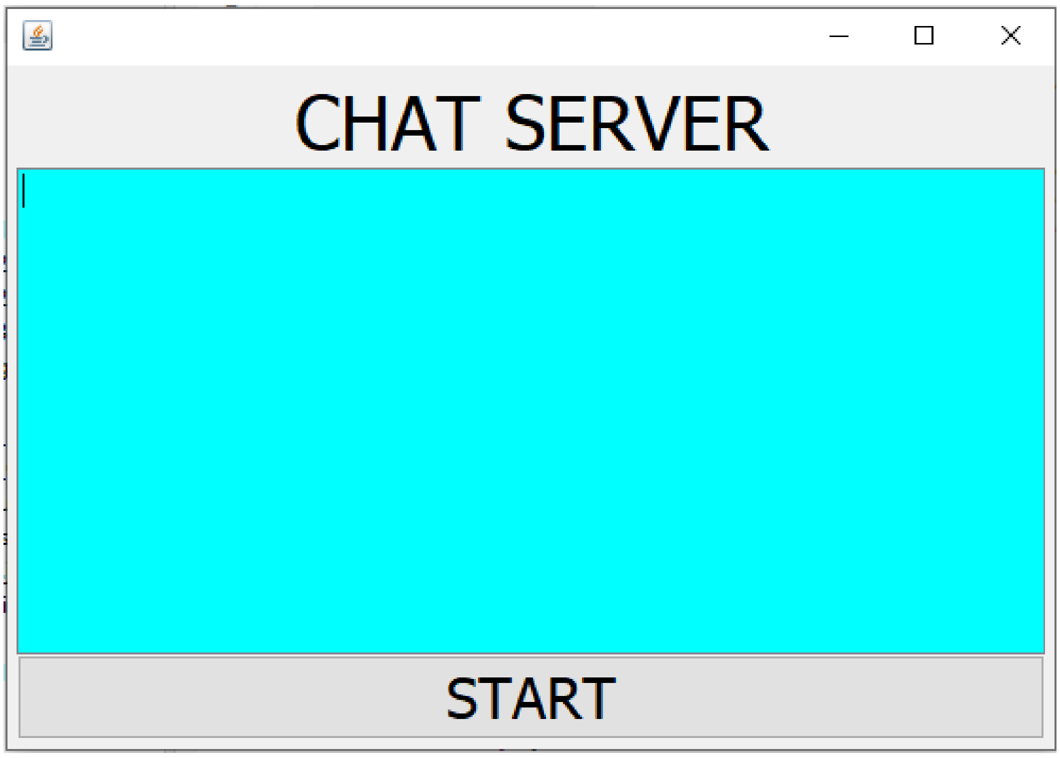


<<Java Class>>
ChatServerCLI
(default package)
connectedClients: HashMap<String,PrintWriter>
MAX_CONNECTED: in
PORT: int
verbose: boolean
server: ServerSocket
ChatServerCLI()
broadcastMessage(String):void
start(boolean):void
stop():void
getRandomPort():int
main(String[]):void
<<Java Class>>
ClientHandler
(default package)
socket: Socket
out: PrintWriter
in: BufferedReader
name: String
ClientHandler(Socket)
run():void

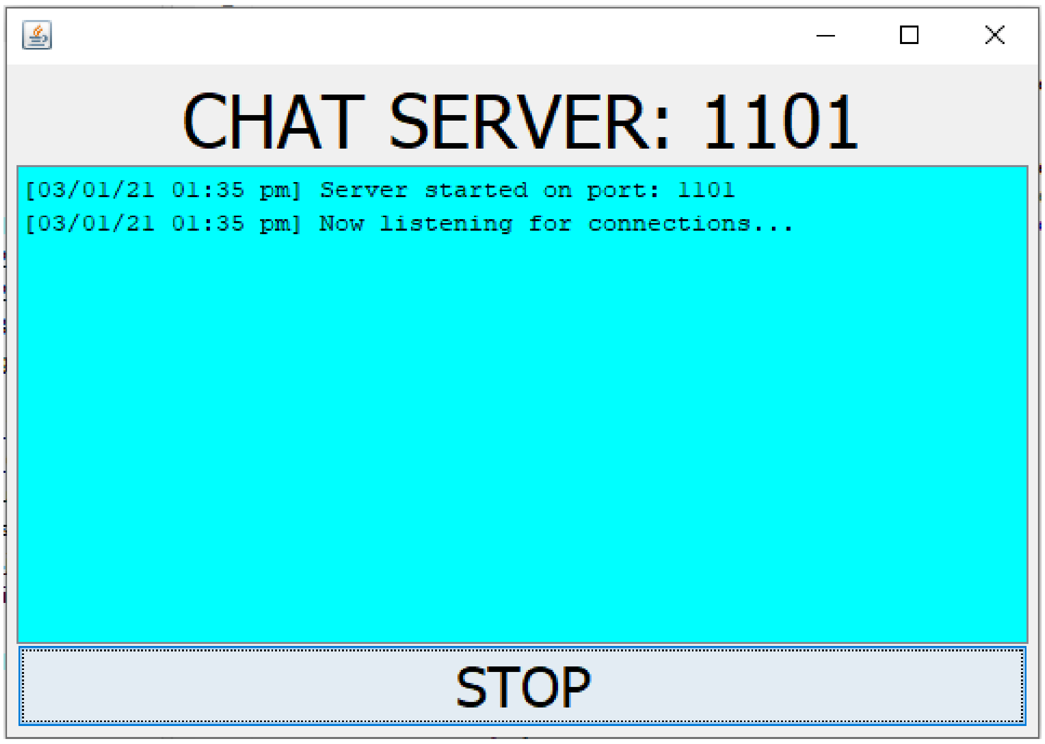

<<Java Class>>
FreePortFinder
me.alexpanov.net
MIN_PORT_NUMBER: in
MAX_PORT_NUMBER: int
LOCK: ServerSocket
currentMinPort: AtomicInteger
FreePortFinder()
findFreeLocalPort():int
findFreeLocalPort(InetAddress):int
findFreeLocalPort(int):int
findFreeLocalPort(int,InetAddress):int
findFreeLocalPortOnAddresses(InetAddress[]):int
available(int):boolean
available(int,InetAddress):boolean

OBSERVATIONS:

Each time we run the class "ServerUI" we get a new server which is capable of handling 50 clients simultaneously.

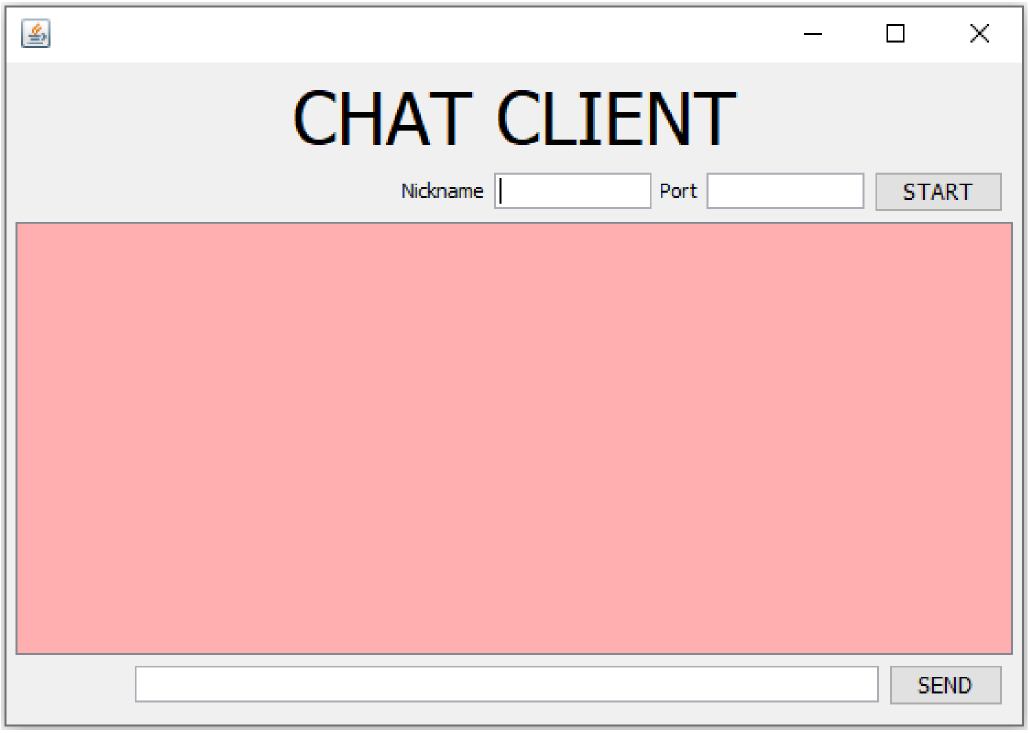


On clicking on the "start" button the server gets enabled and the server gets a unique port number. When a client is in need to be linked to a server, he is supposed to enter port number of the server he wants to connect with.

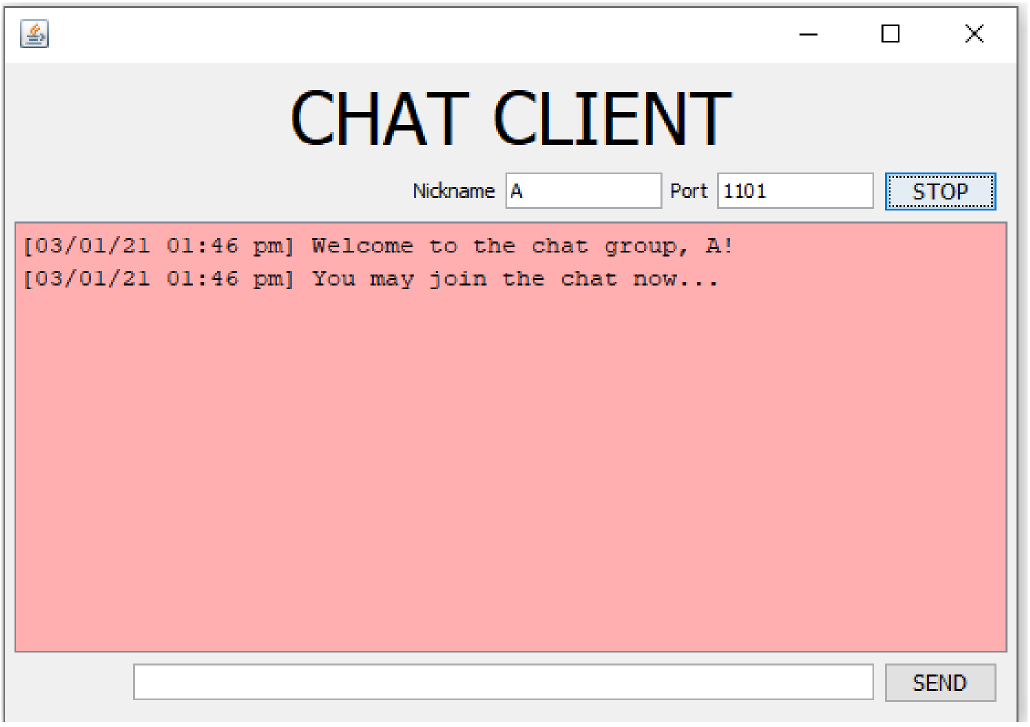


```
[03/01/21 01:35 pm] Server started on port: 1101
[03/01/21 01:35 pm] Now listening for connections...
```

And each time we run the class “ClientUI” we get a new server.

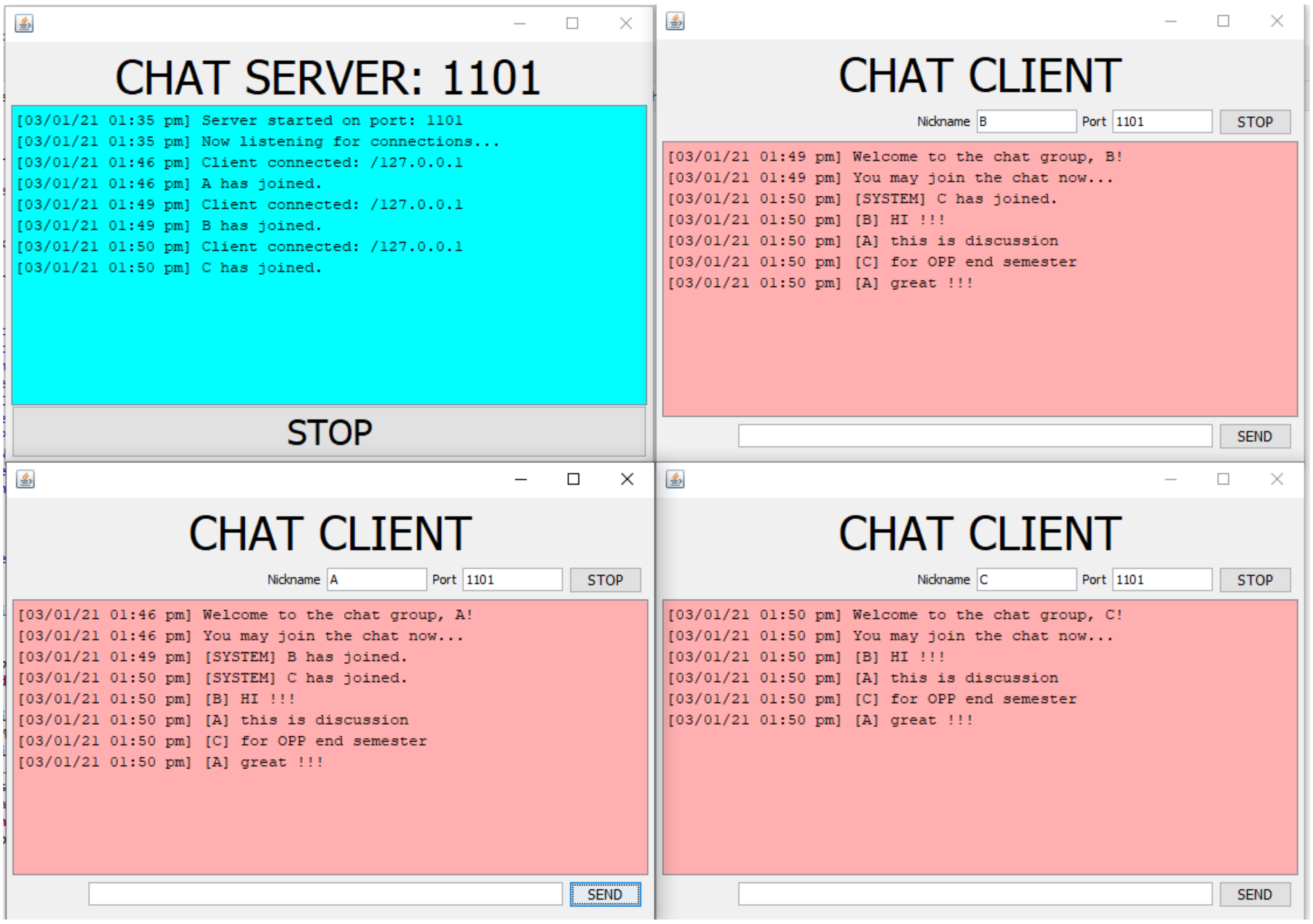


In order to get connected to the server, the user is supposed to enter his nickname and the port number and click the “start” button.



```
[03/01/21 01:46 pm] Welcome to the chat group, A!
[03/01/21 01:46 pm] You may join the chat now...
```

After connecting multiple clients to the server, the messages posted by clients is broadcasted to all the clients connected to the server.



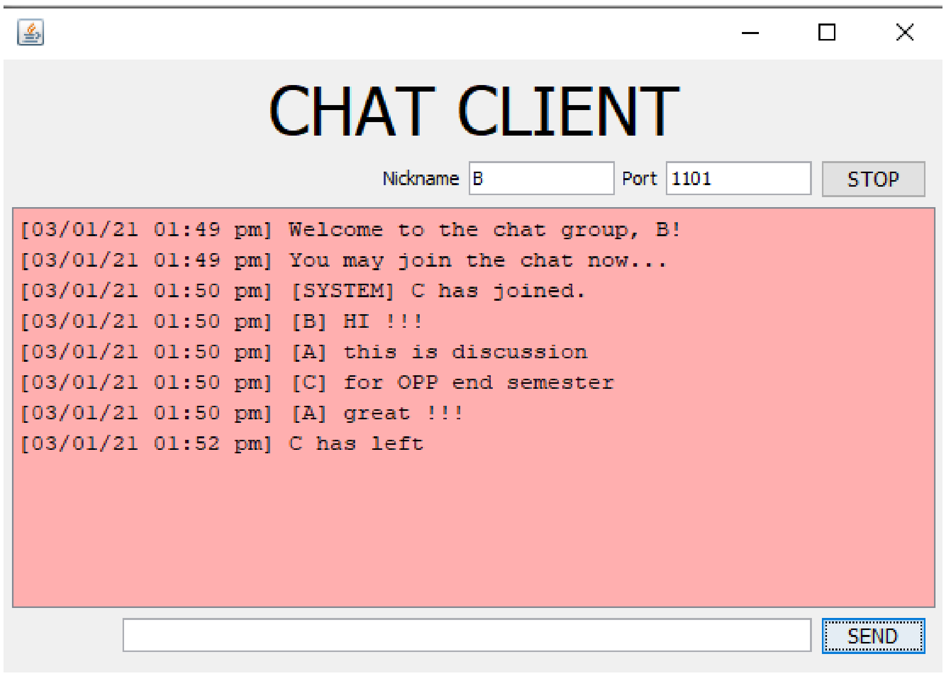
```
[03/01/21 01:35 pm] Server started on port: 1101
[03/01/21 01:35 pm] Now listening for connections...
[03/01/21 01:46 pm] Client connected: /127.0.0.1
[03/01/21 01:46 pm] A has joined.
[03/01/21 01:49 pm] Client connected: /127.0.0.1
[03/01/21 01:49 pm] B has joined.
[03/01/21 01:50 pm] Client connected: /127.0.0.1
[03/01/21 01:50 pm] C has joined.
```

```
[03/01/21 01:49 pm] Welcome to the chat group, B!
[03/01/21 01:49 pm] You may join the chat now...
[03/01/21 01:50 pm] [SYSTEM] C has joined.
[03/01/21 01:50 pm] [B] HI !!!
[03/01/21 01:50 pm] [A] this is discussion
[03/01/21 01:50 pm] [C] for OPP end semester
[03/01/21 01:50 pm] [A] great !!!
```

```
[03/01/21 01:46 pm] Welcome to the chat group, A!
[03/01/21 01:46 pm] You may join the chat now...
[03/01/21 01:49 pm] [SYSTEM] B has joined.
[03/01/21 01:50 pm] [SYSTEM] C has joined.
[03/01/21 01:50 pm] [B] HI !!!
[03/01/21 01:50 pm] [A] this is discussion
[03/01/21 01:50 pm] [C] for OPP end semester
[03/01/21 01:50 pm] [A] great !!!
```

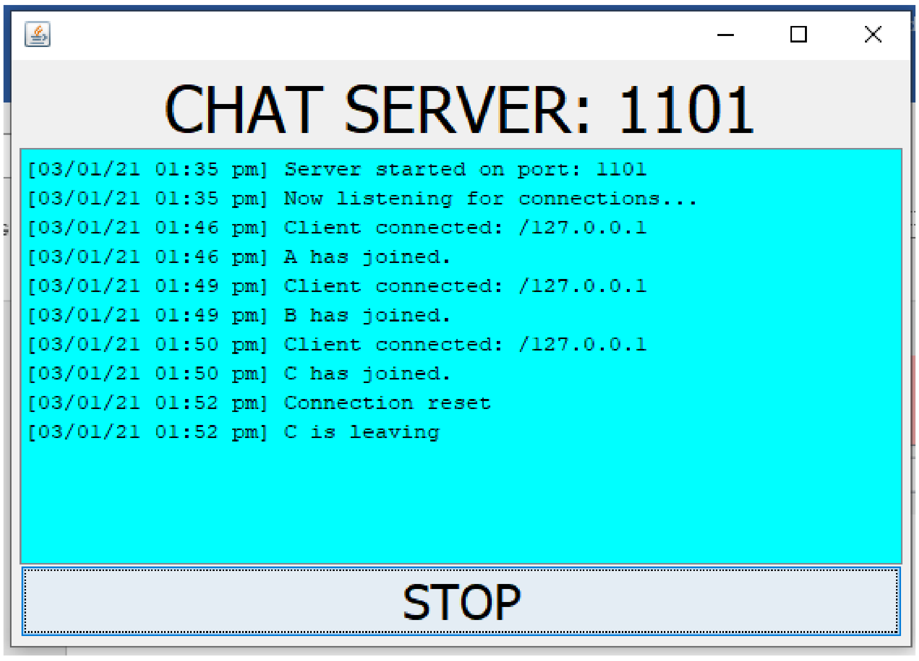
```
[03/01/21 01:50 pm] Welcome to the chat group, C!
[03/01/21 01:50 pm] You may join the chat now...
[03/01/21 01:50 pm] [B] HI !!!
[03/01/21 01:50 pm] [A] this is discussion
[03/01/21 01:50 pm] [C] for OPP end semester
[03/01/21 01:50 pm] [A] great !!!
```

In order to quit from the broadcast, the user can either click the "X" button on the top right corner (or) the user can type "/quit" in the chat.



```
[03/01/21 01:49 pm] Welcome to the chat group, B!
[03/01/21 01:49 pm] You may join the chat now...
[03/01/21 01:50 pm] [SYSTEM] C has joined.
[03/01/21 01:50 pm] [B] HI !!!
[03/01/21 01:50 pm] [A] this is discussion
[03/01/21 01:50 pm] [C] for OPP end semester
[03/01/21 01:50 pm] [A] great !!!
[03/01/21 01:52 pm] C has left
```

If a user quits the broadcast, all the users connected to the broadcast and the server of that broadcast are notified along with the name that one of their clients have left the broadcast.



```
[03/01/21 01:35 pm] Server started on port: 1101
[03/01/21 01:35 pm] Now listening for connections...
[03/01/21 01:46 pm] Client connected: /127.0.0.1
[03/01/21 01:46 pm] A has joined.
[03/01/21 01:49 pm] Client connected: /127.0.0.1
[03/01/21 01:49 pm] B has joined.
[03/01/21 01:50 pm] Client connected: /127.0.0.1
[03/01/21 01:50 pm] C has joined.
[03/01/21 01:52 pm] Connection reset
[03/01/21 01:52 pm] C is leaving
```

CHALLENGES FACED:

- Operates within a computer.
- Unable to maintain chat back-up and clients list back-up.
- Unable to send voice notes
- Unable to exchange files and folders

CONCLUSION:

There are minor improvements and corrections to be made in the project to make it a universally accepted model. The following are the improvements which could make it a universally accepted model:

- Expanding it to operate out of a computer.
- Maintaining a proper back-up of client list and messages.
- Enabling exchange of files and voice-notes
- Strengthening the privacy and security of chats using the cryptographic algorithms