

Socket Based Chat App

BUILT USING JAVA FX

Sudhanshu | Nishant

Contents

Abstract	2
History	2
Technologies Used	3
Java Socket	3
<i>Definition</i>	3
Java Swing	4
<i>History</i>	4
<i>Architecture</i>	5
<i>Configurable</i>	5
<i>Lightweight UI</i>	5
<i>Relationship to AWT</i>	6
<i>Example Screenshot</i>	6
<i>Example Codes</i>	7
<i>Hello World</i>	7
<i>Window with Button</i>	7
Source Code.....	9
Project's working.....	18
Server Welcome Screen.....	18
Client Welcome Screen	19
Client IP Address Dialog Boxoo	19
Client Connected To Server	20
Client's Message Delivery to Server And All Other Connected Clients	21
Server Message Delivery To All Connected Clients	22
Bibliography	23
Github Link To The Project	23
Other Reading Material.....	23

Abstract

Online chat may refer to any kind of communication over the Internet that offers a real-time transmission of text messages from sender to receiver. Chat messages are generally short in order to enable other participants to respond quickly. Thereby, a feeling similar to a spoken conversation is created, which distinguishes chatting from other text-based online communication forms such as Internet forums and email. Online chat may address point-to-point communications as well as multicast communications from one sender to many receivers and voice and video chat, or may be a feature of a web conferencing service.

The term chat room, or chatroom, is primarily used to describe any form of synchronous conferencing, occasionally even asynchronous conferencing. The term can thus mean any technology ranging from real-time online chat and online interaction with strangers (e.g., online forums) to fully immersive graphical social environments.

The primary use of a chat room is to share information via text with a group of other users. Generally speaking, the ability to converse with multiple people in the same conversation differentiates chat rooms from instant messaging programs, which are more typically designed for one-to-one communication. The users in a particular chat room are generally connected via a shared internet or other similar connection, and chat rooms exist catering for a wide range of subjects. New technology has enabled the use of file sharing and webcams to be included in some programs. This would be considered a chat room.

HISTORY

The first online chat system was called Talkomatic, created by Doug Brown and David R. Woolley in 1973 on the PLATO System at the University of Illinois. It offered several channels, each of which could accommodate up to five people, with messages appearing on all users' screens character-by-character as they were typed. Talkomatic was very popular among PLATO users into the mid-1980s. In 2014, Brown and Woolley released a web-based version of Talkomatic.

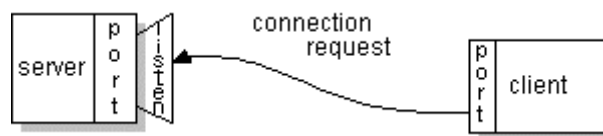
The first online system to use the actual command "chat" was created for The Source in 1979 by Tom Walker and Fritz Thane of Dialcom, Inc.

Technologies Used

JAVA SOCKET

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

Definition

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

The `java.net` package in the Java platform provides a class, `Socket`, that implements one side of a two-way connection between your Java program and another program on the network. The `Socket` class sits on top of a platform-dependent implementation, hiding the details of any particular system from your Java program. By using the `java.net.Socket` class instead of relying on native code, your Java programs can communicate over the network in a platform-independent fashion.

JAVA SWING

Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element.

History

The Internet Foundation Classes (IFC) were a graphics library for Java originally developed by Netscape Communications Corporation and first released on December 16, 1996. On April 2, 1997, Sun Microsystems and Netscape Communications Corporation announced their intention to incorporate IFC with other technologies to form the Java Foundation Classes. The "Java Foundation Classes" were later renamed "Swing."

Swing introduced a mechanism that allowed the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a pluggable look and feel allows Swing components to emulate the appearance of native components while still retaining the benefits of platform independence. Originally distributed as a separately downloadable library, Swing has been included as part of the Java Standard Edition since release 1.2. The Swing classes and components are contained in the `javax.swing` package hierarchy.

Architecture

Swing is a platform-independent, "model-view-controller" GUI framework for Java, which follows a single-threaded programming model. Additionally, this framework provides a layer of abstraction between the code structure and graphic presentation of a Swing-based GUI.

Extensible

Swing is a highly modular-based architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations using Java's inheritance mechanism.

Swing is a component-based framework, whose components are all ultimately derived from the `javax.swing.JComponent` class. Swing objects asynchronously fire events, have bound properties, and respond to a documented set of methods specific to the component. Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

Configurable

Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at run time to fundamental changes in its settings. For example, a Swing-based application is capable of hot swapping its user-interface during runtime. Furthermore, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

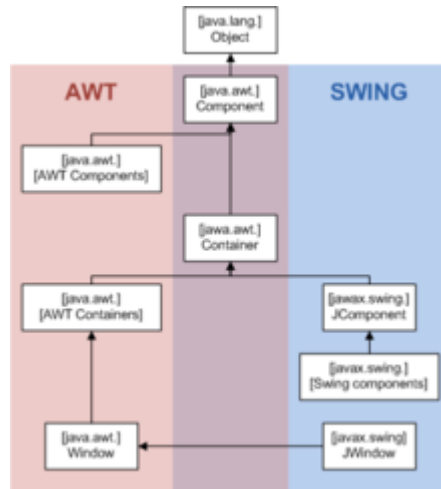
Lightweight UI

Swing's high level of flexibility is reflected in its inherent ability to override the native host operating system (OS)'s GUI controls for displaying itself. Swing "paints" its controls using the Java 2D APIs, rather than calling a native user interface toolkit. Thus, a Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics GUIs.

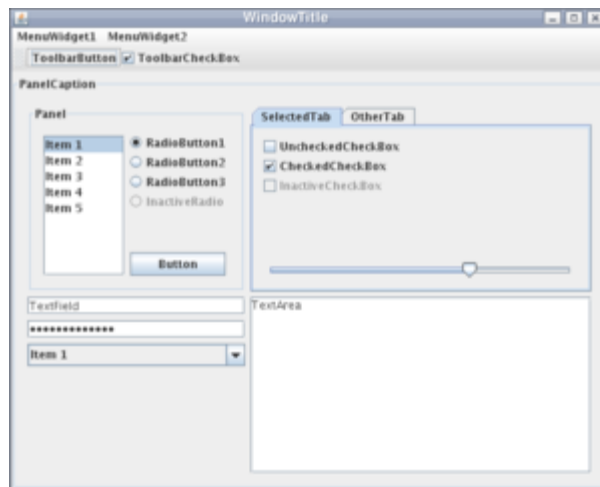
Relationship to AWT

Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system.

By contrast, Swing components are often described as lightweight because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as heavyweight components.



Example Screenshot



Example Codes

Hello World

```
// Hello.java (Java SE 5)
import javax.swing.*;

public class Hello extends JFrame {
    public Hello() {
        super("hello");
        super.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        super.add(new JLabel("Hello, world!"));
        super.pack();
        super.setVisible(true);
    }

    public static void main(final String[] args) {
        new Hello();
    }
}
```

Window with Button

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import javax.swing.SwingUtilities;

public class SwingExample implements Runnable {

    @Override
    public void run() {
        // Create the window
        JFrame f = new JFrame("Hello, !");
        // Sets the behavior for when the window is closed
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        // Add a layout manager so that the button is not placed on top of
the label
        f.setLayout(new FlowLayout());
        // Add a label and a button
        f.add(new JLabel("Hello, world!"));
        f.add(new JButton("Press me!"));
        // Arrange the components inside the window
        f.pack();
    }
}
```



```
        // By default, the window is not visible. Make it visible.  
        f.setVisible(true);  
    }  
  
    public static void main(String[] args) {  
        SwingExample se = new SwingExample();  
        // Schedules the application to be run at the correct time in the  
        event queue.  
        SwingUtilities.invokeLater(se);  
    }  
}
```



Source Code

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package chatappoops;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;

/**
 *
 * @author Nishant Agrawal
 */
public class ClientServer extends javax.swing.JFrame implements Runnable{
    Socket client;
    ServerSocket server;
    BufferedReader Server_Reader,Client_Reader;
    BufferedWriter Server_Writer,Client_Writer;
    /**
     * Creates new form FrameChat
     */
    public ClientServer() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the
form.
     * WARNING: Do NOT modify this code. The content of this method is
always
     * regenerated by the Form Editor.
     */
}
```

```

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jPanel1 = new javax.swing.JPanel();
    jCBserver = new javax.swing.JComboBox<>();
    jBon = new javax.swing.JButton();
    jBsend = new javax.swing.JButton();
    jTusername = new javax.swing.JTextField();
    Lchat = new java.awt.List();
    jTchat = new javax.swing.JTextField();
    jBabout = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jCBserver.setModel(new javax.swing.DefaultComboBoxModel<>(new
String[] { "SERVER", "CLIENT" }));
    jCBserver.addItemListener(new java.awt.event.ItemListener() {
        public void itemStateChanged(java.awt.event.ItemEvent evt) {
            jCBserverItemStateChanged(evt);
        }
    });
    jCBserver.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jCBserverActionPerformed(evt);
        }
    });

    jBon.setText("ON");
    jBon.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jBonActionPerformed(evt);
        }
    });

    jBsend.setText("Send");
    jBsend.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            jBsendActionPerformed(evt);
        }
    });

    jTusername.setText("Username");

```

```

jTchat.setText("Text Message");

jBabout.setText("About");
jBabout.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jBaboutActionPerformed(evt);
    }
});

javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
        .addGap()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
    .addComponent(jTusername)
    .addComponent(Lchat,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addComponent(jCBserver,
javax.swing.GroupLayout.PREFERRED_SIZE, 91,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(70, 70, 70)
            .addComponent(jBon,
javax.swing.GroupLayout.PREFERRED_SIZE, 172,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 106,
Short.MAX_VALUE)
            .addComponent(jBabout,
javax.swing.GroupLayout.PREFERRED_SIZE, 74,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGroup(jPanel1Layout.createSequentialGroup()
            .addComponent(jTchat)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

```

```

        .addComponent(jBsend,
javax.swing.GroupLayout.PREFERRED_SIZE, 72,
javax.swing.GroupLayout.PREFERRED_SIZE)))
        .addContainerGap()
    );
    jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.CENTER)
        .addComponent(jCBserver,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jBabout)
        .addComponent(jBon,
javax.swing.GroupLayout.PREFERRED_SIZE, 23,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(jTusername,
javax.swing.GroupLayout.PREFERRED_SIZE, 28,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addComponent(Lchat,
javax.swing.GroupLayout.PREFERRED_SIZE, 205,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
        .addComponent(jTchat,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
        .addComponent(jBsend))
        .addContainerGap()
    );

```

```

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(10, 10, 10))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jPanel2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(10, 10, 10))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGap(10, 10, 10))
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(jPanel2,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGap(10, 10, 10))
        );

        pack();
    } // </editor-fold>

    private void client_connection(){
        try{
            String ip = JOptionPane.showInputDialog("IP Address: ");
            client = new Socket(ip, 2000);
            jCBserver.setEnabled(false);
            Server_Reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            Server_Writer = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
            jBon.setText("DISCONNECT");
        } catch(UnknownHostException ex){
            System.out.println("Accept Failed");
            System.exit(-1);
        } catch(IOException ex){

```

```

Logger.getLogger(ClientServer.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

private void read_connection(){
    try{
        try{
            try{
                server = new ServerSocket(2000);
                this.setTitle("Please wait...");
            } catch (Exception ex){
                System.out.println("Could not listen");
                System.exit(-1);
            }
            client = server.accept();
            this.setTitle("CONNECTED " + client.getInetAddress());
        } catch (IOException ex){
            System.out.println("Accept Failed");
            System.exit(-1);
        }
        Server_Reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
        Server_Writer = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
        } catch (IOException ex){
            System.out.println("Read Failed");
            System.exit(-1);
        }
    }

private void disconnected_by_client(){
    try{
        client.close();
        Server_Reader.close();
        Server_Writer.close();
        jCBserver.setEnabled(true);
        jBon.setText("CONNECT");
    } catch (IOException ex){

Logger.getLogger(ClientServer.class.getName()).log(Level.SEVERE, null,
ex);
    }
}

```

```

        private void stopped_by_server(){
            try{
                Server_Reader.close();
                Server_Writer.close();
                jBon.setText("ON");
                setTitle("DISCONNECT");
            } catch(IOException ex){

Logger.getLogger(ClientServer.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }

        private void jBonActionPerformed(java.awt.event.ActionEvent evt) {
            if(jBon.getText().equals("CONNECT")){
                jBon.setText("DISCONNECT");
                client_connection();
                Thread thread = new Thread(this);
                thread.start();
            } else if (jCBserver.getSelectedItem().equals("SERVER")){
                jBon.setText("OFF");
                read_connection();
                Thread thread = new Thread(this);
                thread.start();
            }
        }

        private void jBaboutActionPerformed(java.awt.event.ActionEvent evt) {
            // TODO add your handling code here:
        }

        private void jCBserverActionPerformed(java.awt.event.ActionEvent evt)
        {

        }

        private void jBsendActionPerformed(java.awt.event.ActionEvent evt) {
            try{
                Server_Writer.write(jTusername.getText() + ": " +
jTchat.getText());
                Server_Writer.newLine();
                Server_Writer.flush();
            } catch(Exception ex){

```



```

Logger.getLogger(ClientServer.class.getName()).log(Level.SEVERE, null,
ex);
    }
    Lchat.add("Me: " + jTchat.getText());
    jTchat.setText("");
}

private void jCBserverItemStateChanged(java.awt.event.ItemEvent evt) {
    if(jCBserver.getSelectedItem().equals("SERVER")){
        jBon.setText("ON");
        jTusername.setText("SERVER");
    } else{
        jBon.setText("CONNECT");
        jTusername.setText("CLIENT");
    }
}

public static void main(String args[]){
    java.awt.EventQueue.invokeLater(new Runnable(){
        public void run(){
            new ClientServer().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private java.awt.List Lchat;
private javax.swing.JButton jBabout;
private javax.swing.JButton jBon;
private javax.swing.JButton jBsend;
private javax.swing.JComboBox<String> jCBserver;
private javax.swing.JPanel jPanel1;
private javax.swing.JTextField jTchat;
private javax.swing.JTextField jTusername;
// End of variables declaration

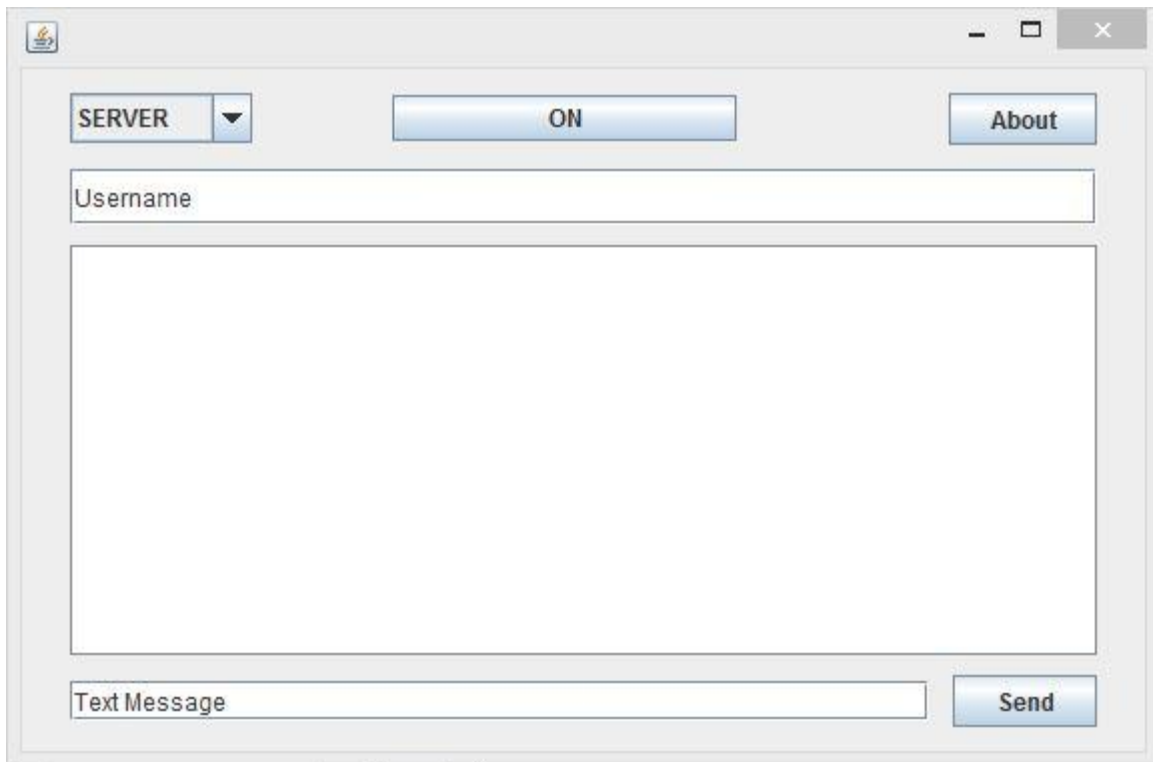
@Override
public void run() {
    while(true){
        try{
            Lchat.add(Server_Reader.readLine());
        } catch(IOException ex){

```

```
Logger.getLogger(ClientServer.class.getName()).log(Level.SEVERE, null,
ex);
    }
}
}
```

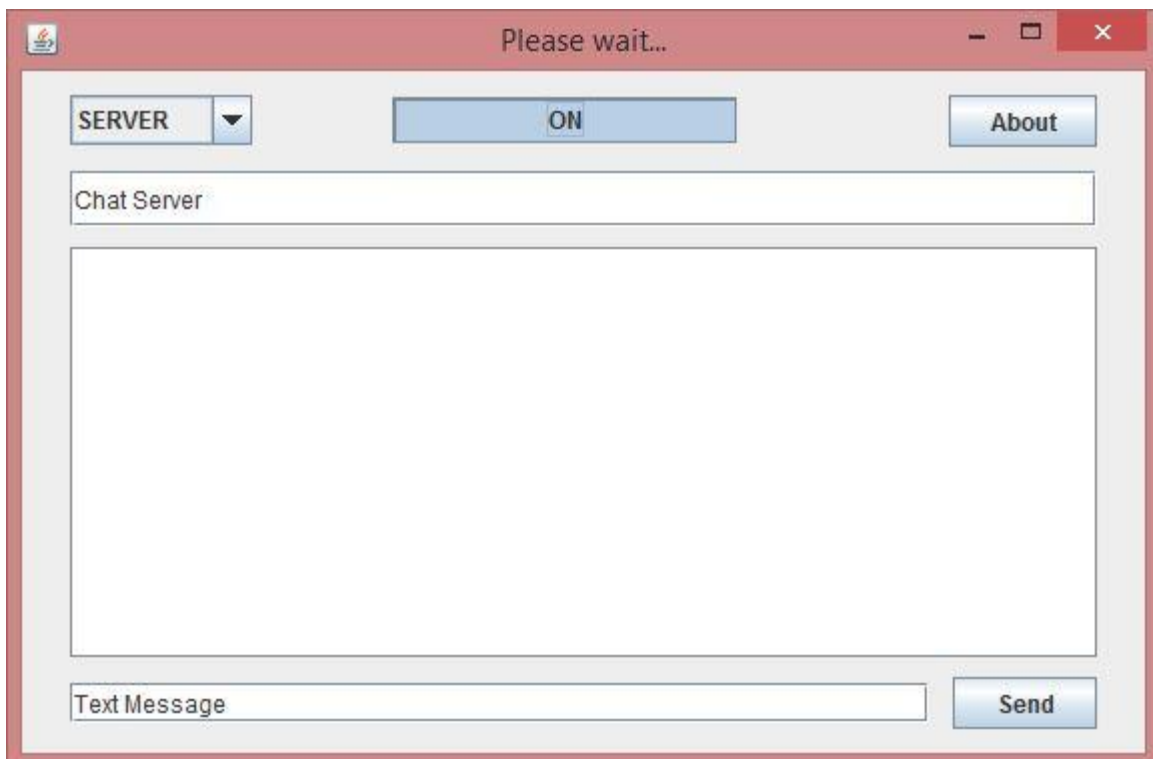
Project's working

SERVER WELCOME SCREEN



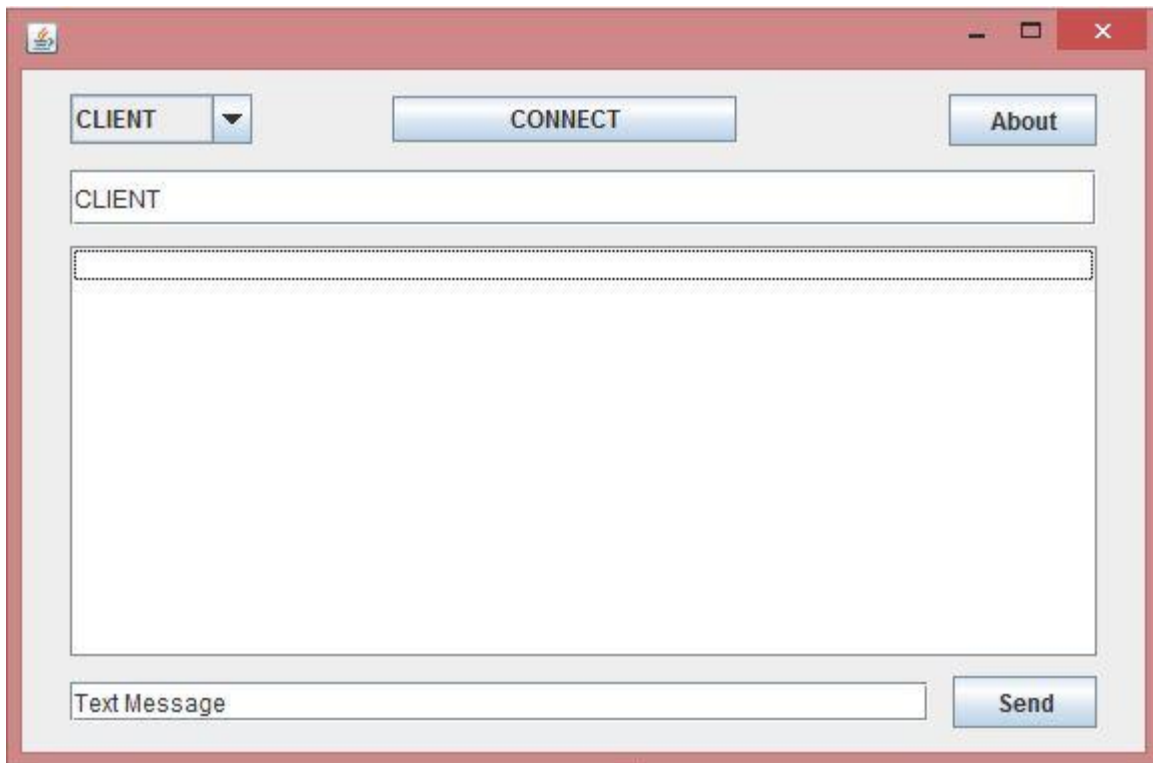
A screenshot of a Windows application window titled "SERVER WELCOME SCREEN". The window has a standard Windows title bar with minimize, maximize, and close buttons. The interface includes a dropdown menu labeled "SERVER" with a downward arrow, a button labeled "ON", and a button labeled "About". Below these is a text input field containing the text "Username". A large, empty rectangular area occupies the center of the window. At the bottom, there is a text input field labeled "Text Message" and a button labeled "Send".

SERVER LISTENING FOR CONNECTION

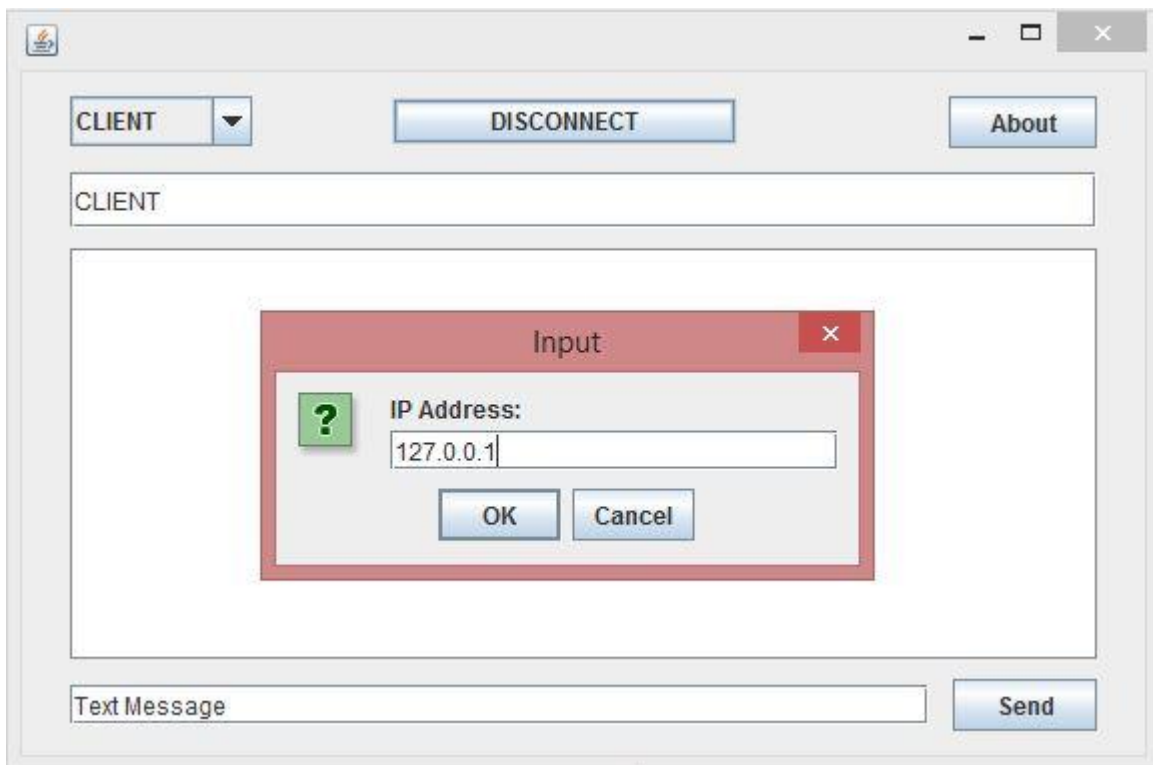


A screenshot of a Windows application window titled "Please wait...". The window has a red title bar, indicating it is a system dialog or a warning. The interface is identical to the previous window, featuring a dropdown menu labeled "SERVER", a button labeled "ON", and a button labeled "About". The text input field now contains the text "Chat Server". The large central area remains empty. At the bottom, there is a text input field labeled "Text Message" and a button labeled "Send".

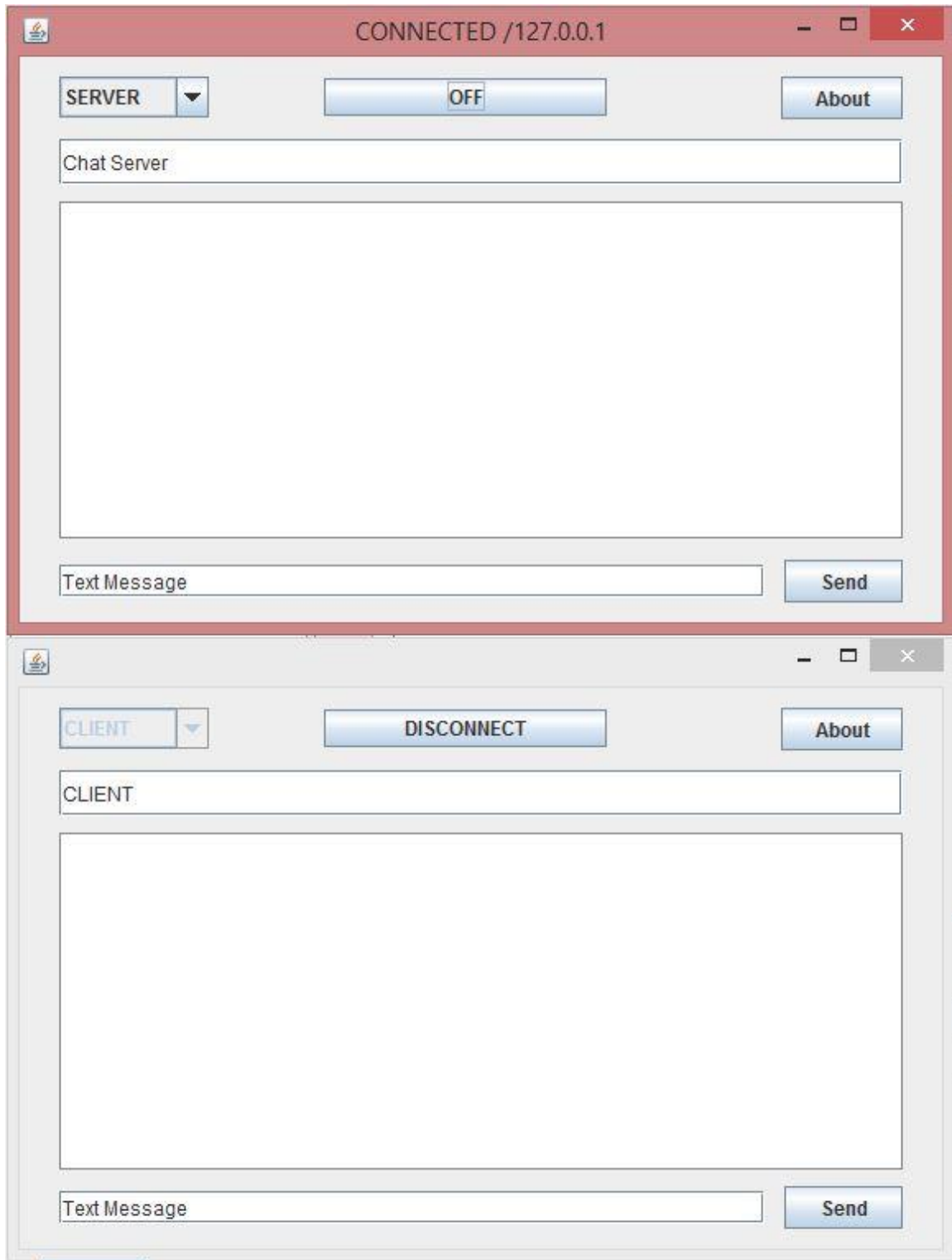
CLIENT WELCOME SCREEN



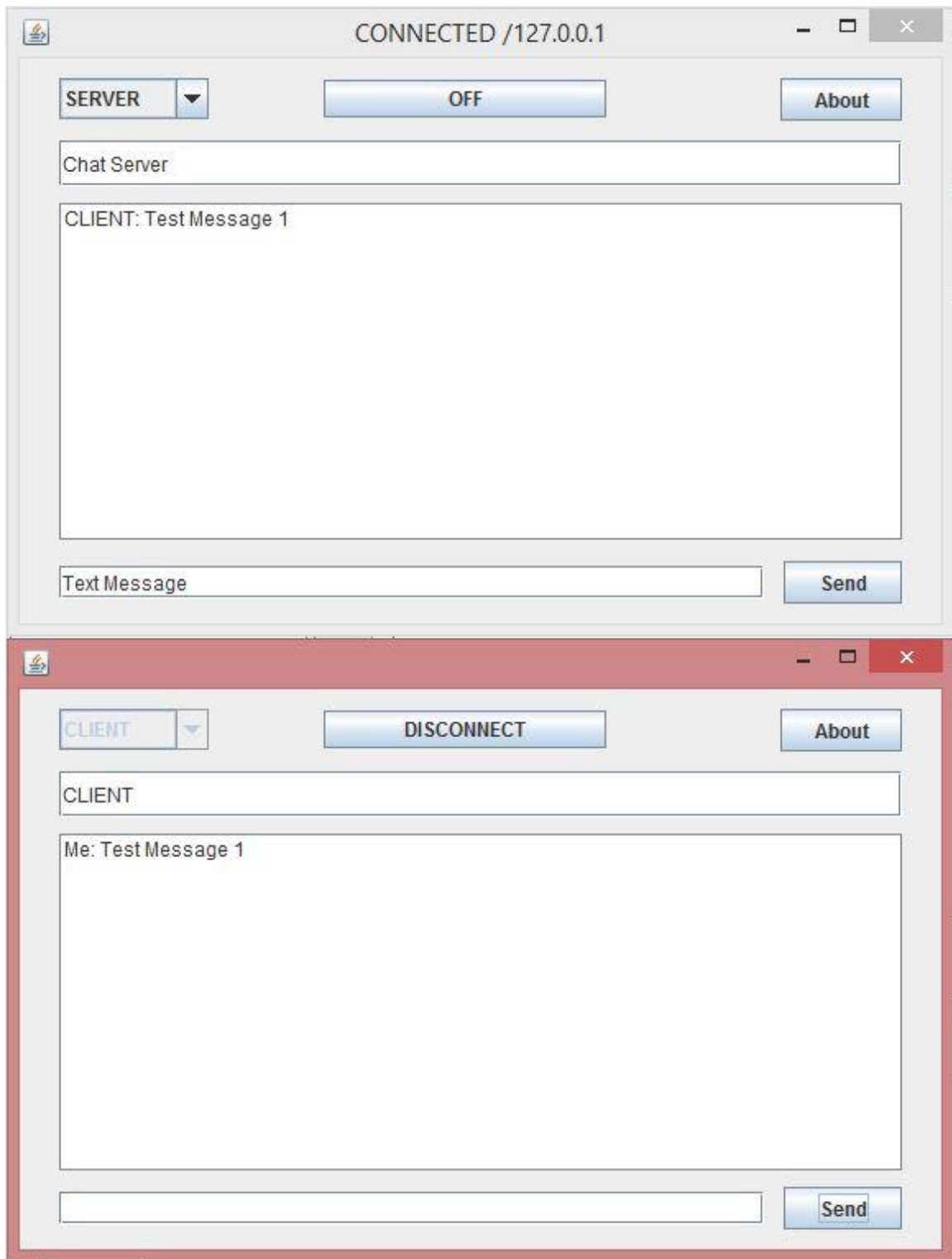
CLIENT IP ADDRESS DIALOG BOX



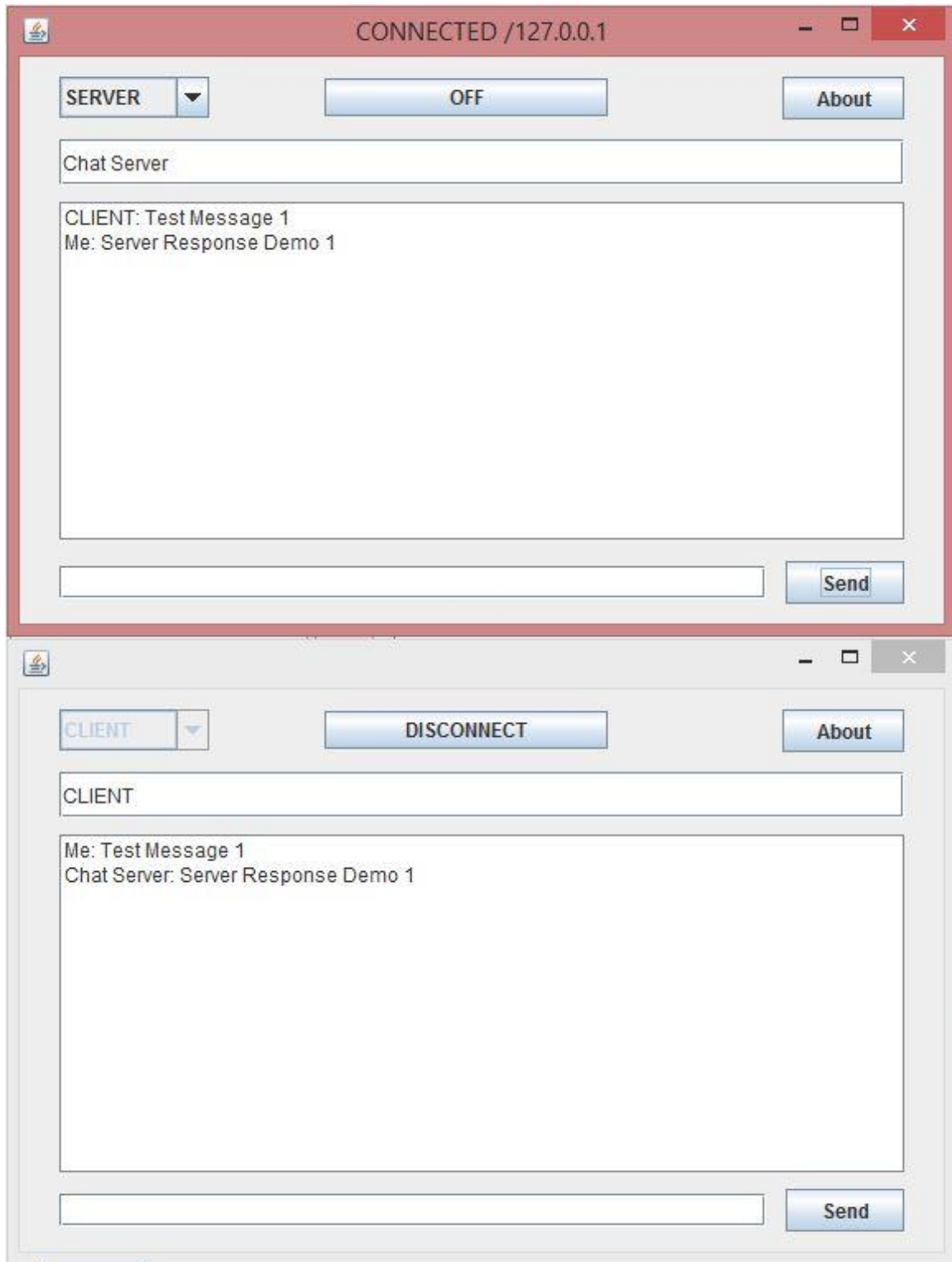
CLIENT CONNECTED TO SERVER



CLIENT'S MESSAGE DELIVERY TO SERVER AND ALL OTHER CONNECTED CLIENTS



SERVER MESSAGE DELIVERY TO ALL CONNECTED CLIENTS



Bibliography

GITHUB LINK TO THE PROJECT

<https://github.com/nishantoo13/Chat-App>

OTHER READING MATERIAL

API Documentation of Socket:

<https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html>

Socket Tutorial:

<https://docs.oracle.com/javase/tutorial/networking/sockets/>

Java Swing Briefing:

[https://en.wikipedia.org/wiki/Swing_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java))

Java Swing API Documentation:

<https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

Java Swing Tutorial:

<https://www.javatpoint.com/java-swing>