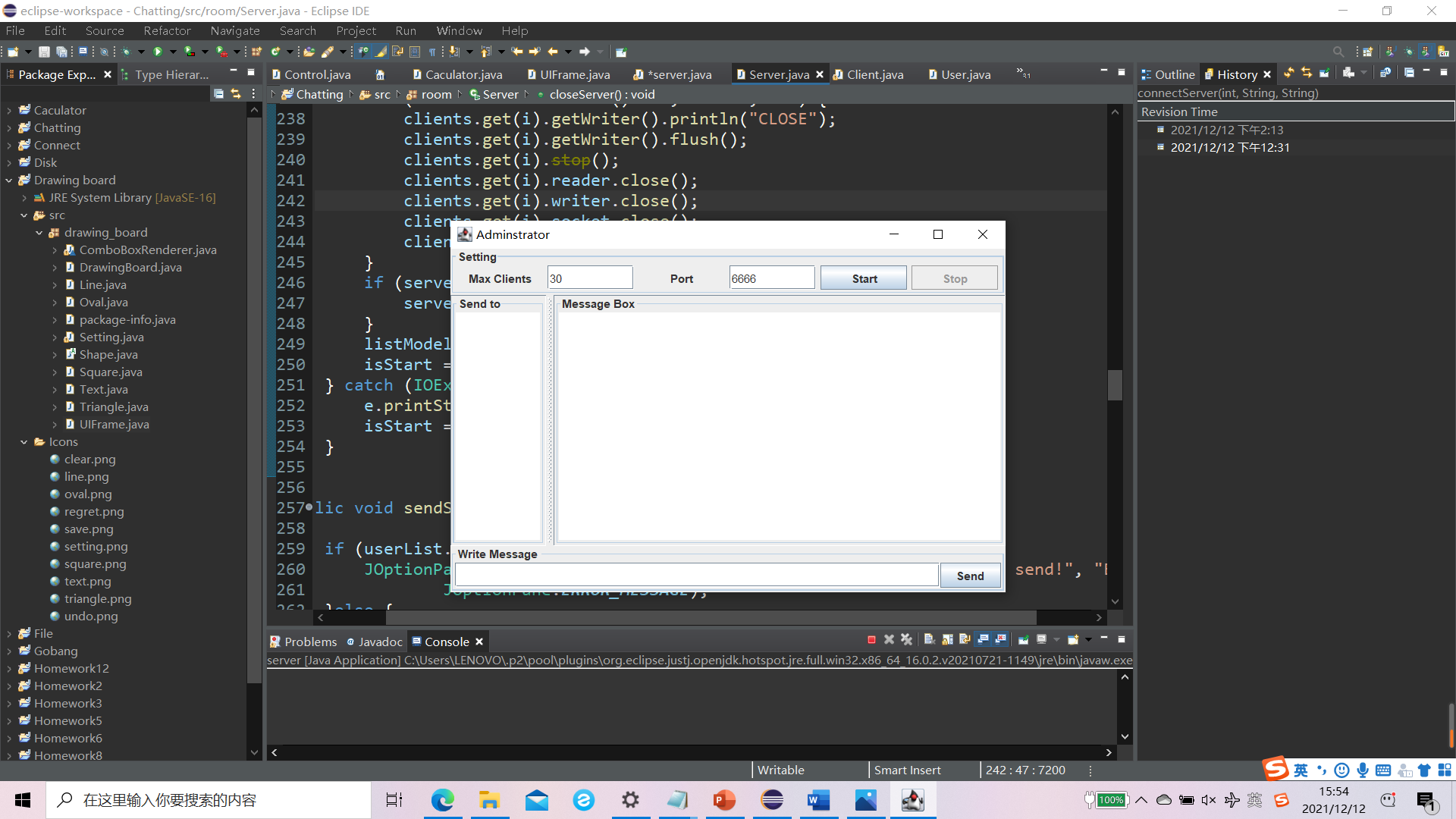
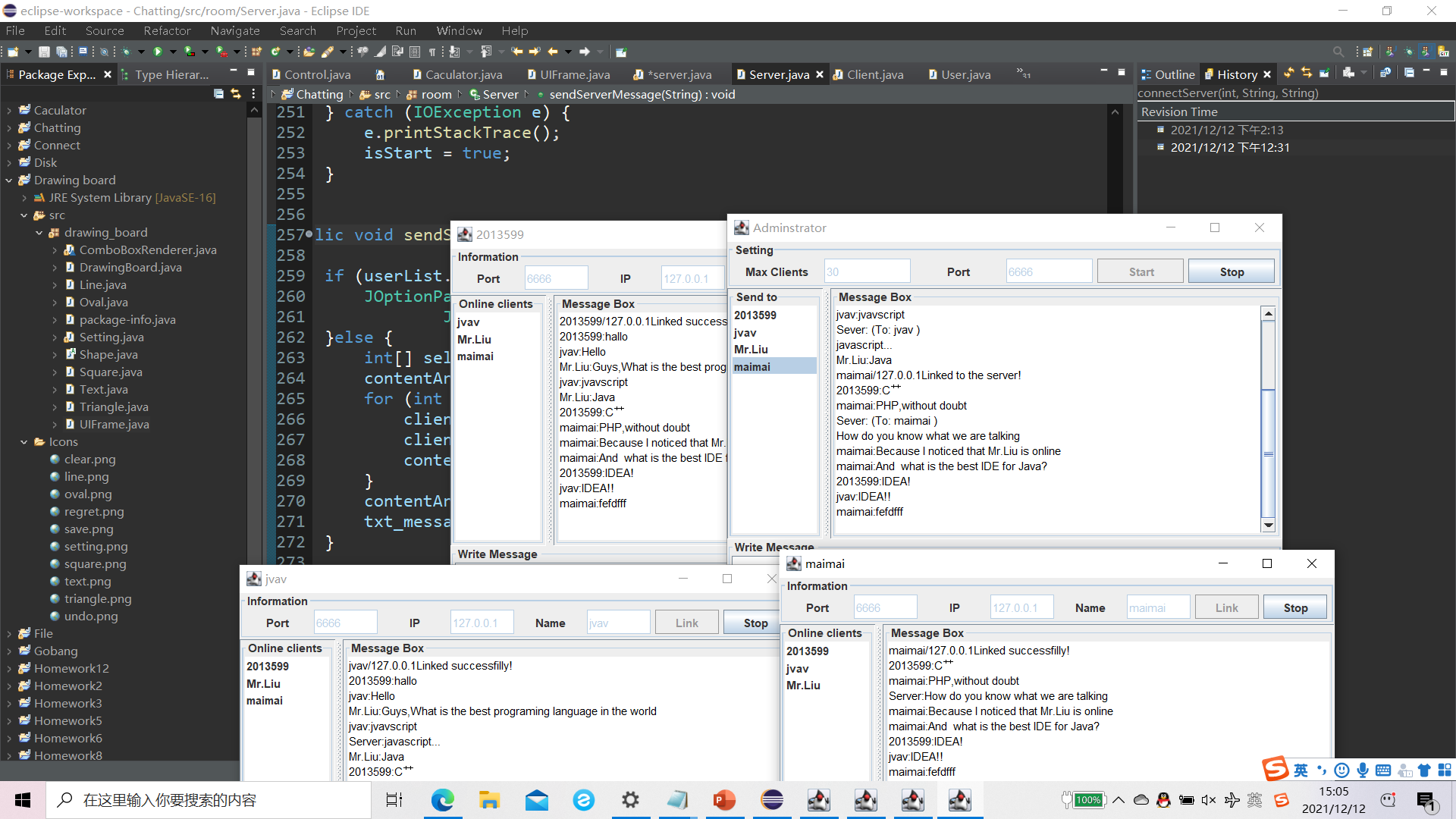
Java网络编程作业

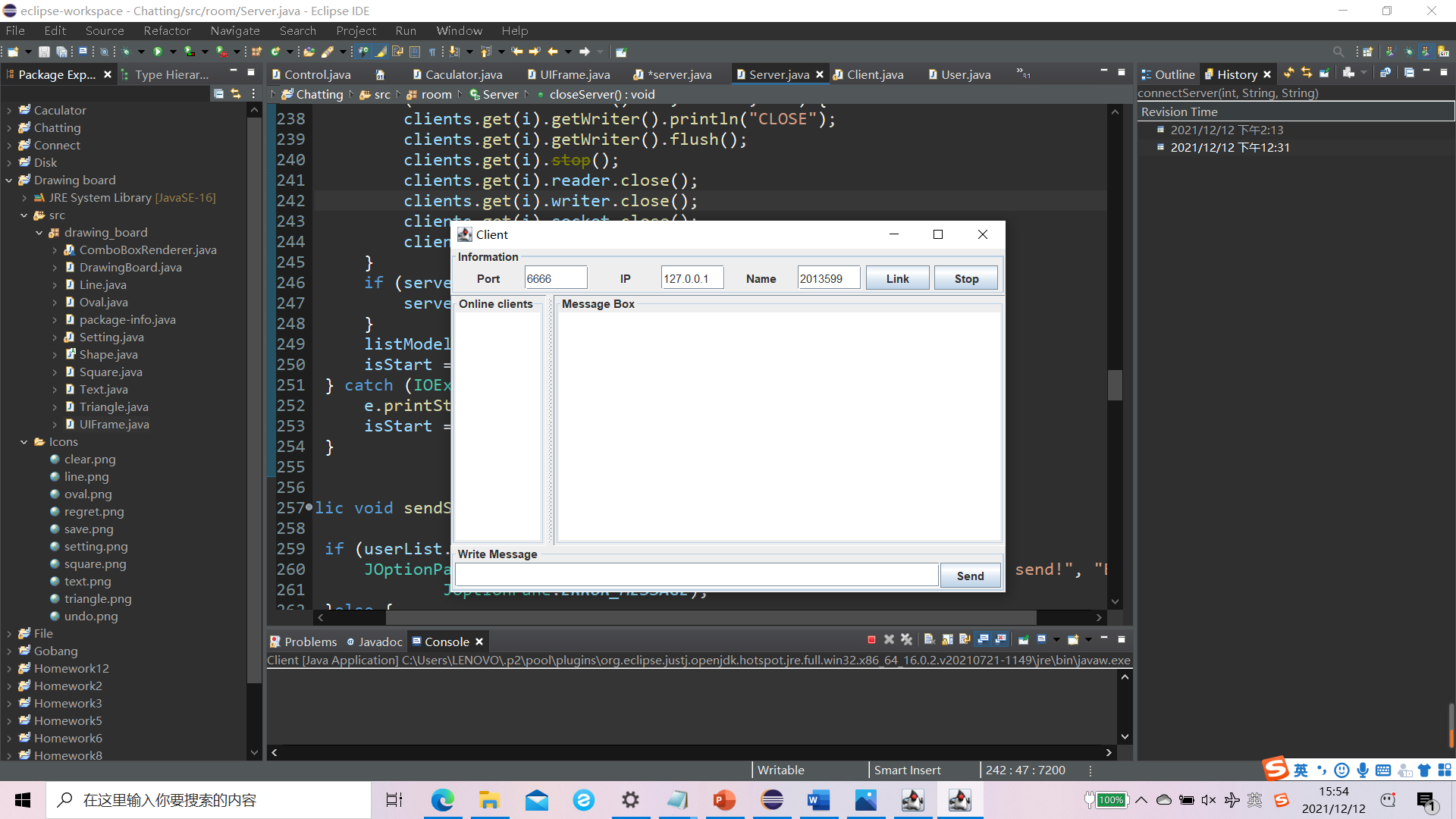
2013599 田佳业

一、程序亮点:

对通信程序的页面进行优化，同时拓展功能，增加了最大限制人数，利用JList优化了选择发送者的方式，初步实现了多人聊天功能，为后续的大作业程序做准备。

二、运行实例:





三、核心代码：

Server.java

package room;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.GridLayout;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.BindException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.StringTokenizer;

import javax.swing.DefaultListModel;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JSplitPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import javax.swing.SwingConstants;

import javax.swing.border.TitledBorder;

public class Server {

    private JFrame frame;

    private JTextArea contentArea;

    private JTextField txt\_message;

    private JTextField txt\_max;

    private JTextField txt\_port;

    private JButton btn\_start;

    private JButton btn\_stop;

    private JButton btn\_send;

    private JPanel northPanel;

    private JPanel southPanel;

    private JScrollPane rightPanel;

    private JScrollPane leftPanel;

    private JSplitPane centerSplit;

    private JList<String> userList;

    private DefaultListModel<String> listModel;

    private ServerSocket serverSocket;

    private ServerThread serverThread;

    private ArrayList<ClientThread> clients;

    private boolean isStart = false;

    public static void main(String[] args) {

        new Server();

    }

    public void send() {

        if (!isStart) {

            JOptionPane.showMessageDialog(frame, "Please start the server!", "Error",

                    JOptionPane.ERROR\_MESSAGE);

            return;

        }

        if (clients.size() == 0) {

            JOptionPane.showMessageDialog(frame, "No user online", "Warning",

                    JOptionPane.WARNING\_MESSAGE);

            return;

        }

        String message = txt\_message.getText().trim();

        if (message == null || message.equals("")) {

            JOptionPane.showMessageDialog(frame, "Empty message", "Error",

                    JOptionPane.ERROR\_MESSAGE);

            return;

        }

        sendServerMessage(message);

    }

    public Server() {

        frame = new JFrame("Adminstrator");

        contentArea = new JTextArea();

        contentArea.setEditable(false);

        contentArea.setForeground(Color.black);

        txt\_message = new JTextField();

        txt\_max = new JTextField("30");

        txt\_port = new JTextField("6666");

        btn\_start = new JButton("Start");

        btn\_stop = new JButton("Stop");

        btn\_send = new JButton("Send");

        btn\_stop.setEnabled(false);

        listModel = new DefaultListModel<String>();

        userList = new JList<String>(listModel);

        southPanel = new JPanel(new BorderLayout());

        southPanel.setBorder(new TitledBorder("Write Message"));

        southPanel.add(txt\_message, "Center");

        southPanel.add(btn\_send, "East");

        leftPanel = new JScrollPane(userList);

        leftPanel.setBorder(new TitledBorder("Send to"));

        rightPanel = new JScrollPane(contentArea);

        rightPanel.setBorder(new TitledBorder("Message Box"));

        centerSplit = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT, leftPanel,

                rightPanel);

        centerSplit.setDividerLocation(100);

        northPanel = new JPanel();

        northPanel.setLayout(new GridLayout(1, 6,5,0));

        northPanel.add(new JLabel("Max Clients",SwingConstants.CENTER));

        northPanel.add(txt\_max);

        northPanel.add(new JLabel("Port",SwingConstants.CENTER));

        northPanel.add(txt\_port);

        northPanel.add(btn\_start);

        northPanel.add(btn\_stop);

        northPanel.setBorder(new TitledBorder("Setting"));

        frame.setLayout(new BorderLayout());

        frame.add(northPanel, "North");

        frame.add(centerSplit, "Center");

        frame.add(southPanel, "South");

        frame.setSize(600, 400);

        int screen\_width = Toolkit.getDefaultToolkit().getScreenSize().width;

        int screen\_height = Toolkit.getDefaultToolkit().getScreenSize().height;

        frame.setLocation((screen\_width - frame.getWidth()) / 2,

                (screen\_height - frame.getHeight()) / 2);

        frame.setVisible(true);

        frame.addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent e) {

                if (isStart) {

                    closeServer();

                }

                System.exit(0);

            }

        });

        txt\_message.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                send();

            }

        });

        btn\_send.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent arg0) {

                send();

            }

        });

        btn\_start.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if (isStart) {

                    JOptionPane.showMessageDialog(frame, "Already Started",

                            "Error", JOptionPane.ERROR\_MESSAGE);

                    return;

                }

                int max;

                int port;

                try {

                    try {

                        max = Integer.parseInt(txt\_max.getText());

                    } catch (Exception e1) {

                        throw new Exception("Format error!");

                    }

                    if (max <= 0) {

                        throw new Exception("Format error!");

                    }

                    try {

                        port = Integer.parseInt(txt\_port.getText());

                    } catch (Exception e1) {

                        throw new Exception("Format error!");

                    }

                    if (port <= 0) {

                        throw new Exception("Format error!");

                    }

                    serverStart(max, port);

                    contentArea.append("Start successfully!\nMax client:" + max + ",Port：" + port

                            + "\r\n");

                    btn\_start.setEnabled(false);

                    txt\_max.setEnabled(false);

                    txt\_port.setEnabled(false);

                    btn\_stop.setEnabled(true);

                } catch (Exception exc) {

                    JOptionPane.showMessageDialog(frame, exc.getMessage(),

                            "Error", JOptionPane.ERROR\_MESSAGE);

                }

            }

        });

        btn\_stop.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if (!isStart) {

                    return;

                }

                try {

                    closeServer();

                    btn\_start.setEnabled(true);

                    txt\_max.setEnabled(true);

                    txt\_port.setEnabled(true);

                    btn\_stop.setEnabled(false);

                    contentArea.append("Close successfully!\r\n");

                    JOptionPane.showMessageDialog(frame, "Close successfully!");

                } catch (Exception exc) {

                    JOptionPane.showMessageDialog(frame, "Error！", "Error",

                            JOptionPane.ERROR\_MESSAGE);

                }

            }

        });

    }

    public void serverStart(int max, int port) throws java.net.BindException {

        try {

            clients = new ArrayList<ClientThread>();

            serverSocket = new ServerSocket(port);

            serverThread = new ServerThread(serverSocket, max);

            serverThread.start();

            isStart = true;

        } catch (BindException e) {

            isStart = false;

            throw new BindException("This Port is used by aother thread!");

        } catch (Exception e1) {

            e1.printStackTrace();

            isStart = false;

        }

    }

    @SuppressWarnings("deprecation")

    public void closeServer() {

        try {

            if (serverThread != null)

                serverThread.stop();

            for (int i = clients.size() - 1; i >= 0; i--) {

                clients.get(i).getWriter().println("CLOSE");

                clients.get(i).getWriter().flush();

                clients.get(i).stop();

                clients.get(i).reader.close();

                clients.get(i).writer.close();

                clients.get(i).socket.close();

                clients.remove(i);

            }

            if (serverSocket != null) {

                serverSocket.close();

            }

            listModel.removeAllElements();

            isStart = false;

        } catch (IOException e) {

            e.printStackTrace();

            isStart = true;

        }

    }

    public void sendServerMessage(String message) {

        if (userList.getSelectedIndices().length==0) {

            JOptionPane.showMessageDialog(frame, "Please Choose the client to send!", "Error",

                    JOptionPane.ERROR\_MESSAGE);

        }else {

            int[] selectedUser=userList.getSelectedIndices();

            contentArea.append("Sever: (To: ");

            for (int i =0;i<selectedUser.length;i++) {

                clients.get(selectedUser[i]).getWriter().println("Server:" + message);

                clients.get(selectedUser[i]).getWriter().flush();

                contentArea.append(userList.getSelectedValuesList().get(i)+" ");

            }

            contentArea.append(")\n" + txt\_message.getText() + "\r\n");

            txt\_message.setText(null);

        }

    }

    class ServerThread extends Thread {

        private ServerSocket serverSocket;

        private int max;

        public ServerThread(ServerSocket serverSocket, int max) {

            this.serverSocket = serverSocket;

            this.max = max;

        }

        public void run() {

            while (true) {

                try {

                    Socket socket = serverSocket.accept();

                    if (clients.size() == max) {

                        BufferedReader r = new BufferedReader(

                                new InputStreamReader(socket.getInputStream()));

                        PrintWriter w = new PrintWriter(socket

                                .getOutputStream());

                        String inf = r.readLine();

                        StringTokenizer st = new StringTokenizer(inf, "#");

                        User user = new User(st.nextToken(), st.nextToken());

                        w.println("MAX#Sorry," + user.getName()

                                + user.getIp() + ".is full");

                        w.flush();

                        r.close();

                        w.close();

                        socket.close();

                        continue;

                    }

                    ClientThread client = new ClientThread(socket);

                    client.start();

                    clients.add(client);

                    listModel.addElement(client.getUser().getName());

                    contentArea.append(client.getUser().getName()

                            + client.getUser().getIp() + "Linked to the server!\r\n");

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

    }

    class ClientThread extends Thread {

        private Socket socket;

        private BufferedReader reader;

        private PrintWriter writer;

        private User user;

        public BufferedReader getReader() {

            return reader;

        }

        public PrintWriter getWriter() {

            return writer;

        }

        public User getUser() {

            return user;

        }

        public ClientThread(Socket socket) {

            try {

                this.socket = socket;

                reader = new BufferedReader(new InputStreamReader(socket

                        .getInputStream()));

                writer = new PrintWriter(socket.getOutputStream());

                String inf = reader.readLine();

                StringTokenizer st = new StringTokenizer(inf, "#");

                user = new User(st.nextToken(), st.nextToken());

                writer.println(user.getName() + user.getIp() + "Linked successfilly!");

                writer.flush();

                if (clients.size() > 0) {

                    String temp = "";

                    for (int i =0; i<clients.size(); i++) {

                        temp += (clients.get(i).getUser().getName())

                                + "#";

                    }

                    writer.println("USERLIST#" + clients.size() + "#" + temp);

                    writer.flush();

                }

                for (int i = clients.size() - 1; i >= 0; i--) {

                    clients.get(i).getWriter().println(

                            "ADD#" + user.getName());

                    clients.get(i).getWriter().flush();

                }

            } catch (IOException e) {

                e.printStackTrace();

            }

        }

        @SuppressWarnings("deprecation")

        public void run() {

            String message = null;

            while (true) {

                try {

                    message = reader.readLine();

                    if (message.equals("CLOSE"))

                    {

                        contentArea.append(this.getUser().getName()

                                + this.getUser().getIp() + "Exit!\r\n");

                        reader.close();

                        writer.close();

                        socket.close();

                        for (int i = clients.size() - 1; i >= 0; i--) {

                            clients.get(i).getWriter().println(

                                    "DELETE#" + user.getName());

                            clients.get(i).getWriter().flush();

                        }

                        listModel.removeElement(user.getName());

                        for (int i = clients.size() - 1; i >= 0; i--) {

                            if (clients.get(i).getUser() == user) {

                                ClientThread temp = clients.get(i);

                                clients.remove(i);

                                temp.stop();

                                return;

                            }

                        }

                    } else {

                        dispatcherMessage(message);

                    }

                } catch (IOException e) {

                    e.printStackTrace();

                }

            }

        }

        public void dispatcherMessage(String message) {

            StringTokenizer stringTokenizer = new StringTokenizer(message, "#");

            String source = stringTokenizer.nextToken();

            String owner = stringTokenizer.nextToken();

            String content = stringTokenizer.nextToken();

            message = source + ":" + content;

            contentArea.append(message + "\r\n");

            if (owner.equals("ALL")) {

                for (int i = clients.size() - 1; i >= 0; i--) {

                    clients.get(i).getWriter().println(message);

                    clients.get(i).getWriter().flush();

                }

            }

        }

    }

}

Client.java

package room;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.GridLayout;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import java.util.HashMap;

import java.util.Map;

import java.util.StringTokenizer;

import javax.swing.DefaultListModel;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JList;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JSplitPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import javax.swing.SwingConstants;

import javax.swing.border.TitledBorder;

public class Client{

    private JFrame frame;

    private JList<String> userList;

    private JTextArea textArea;

    private JTextField textField;

    private JTextField txt\_port;

    private JTextField txt\_hostIp;

    private JTextField txt\_name;

    private JButton btn\_start;

    private JButton btn\_stop;

    private JButton btn\_send;

    private JPanel northPanel;

    private JPanel southPanel;

    private JScrollPane rightScroll;

    private JScrollPane leftScroll;

    private JSplitPane centerSplit;

    private DefaultListModel<String> listModel;

    private boolean isConnected = false;

    private Socket socket;

    private PrintWriter writer;

    private BufferedReader reader;

    private MessageThread messageThread;

    public static void main(String[] args) {

        new Client();

    }

    public void send() {

        if (!isConnected) {

            JOptionPane.showMessageDialog(frame, "You have not linked to the sever", "Error",

                    JOptionPane.ERROR\_MESSAGE);

            return;

        }

        String message = textField.getText().trim();

        if (message == null || message.equals("")) {

            JOptionPane.showMessageDialog(frame, "Empty message!", "Warning",

                    JOptionPane.WARNING\_MESSAGE);

            return;

        }

        sendMessage(frame.getTitle() + "#" + "ALL" + "#" + message);

        textField.setText(null);

    }

    public Client() {

        textArea = new JTextArea();

        textArea.setEditable(false);

        textArea.setForeground(Color.black);

        textField = new JTextField();

        txt\_port = new JTextField("6666");

        txt\_hostIp = new JTextField("127.0.0.1");

        txt\_name = new JTextField("2013599");

        btn\_start = new JButton("Link");

        btn\_stop = new JButton("Stop");

        btn\_send = new JButton("Send");

        listModel = new DefaultListModel<String>();

        userList = new JList<String>(listModel);

        northPanel = new JPanel();

        northPanel.setLayout(new GridLayout(1,7,5,0));

        northPanel.add(new JLabel("Port",SwingConstants.CENTER));

        northPanel.add(txt\_port);

        northPanel.add(new JLabel("IP",SwingConstants.CENTER));

        northPanel.add(txt\_hostIp);

        northPanel.add(new JLabel("Name",SwingConstants.CENTER));

        northPanel.add(txt\_name);

        northPanel.add(btn\_start);

        northPanel.add(btn\_stop);

        northPanel.setBorder(new TitledBorder("Information"));

        rightScroll = new JScrollPane(textArea);

        rightScroll.setBorder(new TitledBorder("Message Box"));

        leftScroll = new JScrollPane(userList);

        leftScroll.setBorder(new TitledBorder("Online clients"));

        southPanel = new JPanel(new BorderLayout());

        southPanel.add(textField, "Center");

        southPanel.add(btn\_send, "East");

        southPanel.setBorder(new TitledBorder("Write Message"));

        centerSplit = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT, leftScroll,

                rightScroll);

        centerSplit.setDividerLocation(100);

        frame = new JFrame("Client");

        frame.setLayout(new BorderLayout());

        frame.add(northPanel, "North");

        frame.add(centerSplit, "Center");

        frame.add(southPanel, "South");

        frame.setSize(600, 400);

        int screen\_width = Toolkit.getDefaultToolkit().getScreenSize().width;

        int screen\_height = Toolkit.getDefaultToolkit().getScreenSize().height;

        frame.setLocation((screen\_width - frame.getWidth()) / 2,

                (screen\_height - frame.getHeight()) / 2);

        frame.setVisible(true);

        textField.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent arg0) {

                send();

            }

        });

        btn\_send.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                send();

            }

        });

        btn\_start.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                int port;

                if (isConnected) {

                    return;

                }

                try {

                    try {

                        port = Integer.parseInt(txt\_port.getText().trim());

                    } catch (NumberFormatException e2) {

                        throw new Exception("Format error!");

                    }

                    String hostIp = txt\_hostIp.getText().trim();

                    String name = txt\_name.getText().trim();

                    if (name.equals("") || hostIp.equals("")) {

                        throw new Exception("Cannot be empty!");

                    }

                    boolean flag = connectServer(port, hostIp, name);

                    if (flag == false) {

                        throw new Exception("Connection Failure!");

                    }

                    frame.setTitle(name);

                    JOptionPane.showMessageDialog(frame, "Coneect successfully!  Port:"+txt\_port.getText());

                    btn\_start.setEnabled(false);

                    txt\_port.setEnabled(false);

                    txt\_hostIp.setEnabled(false);

                    txt\_name.setEnabled(false);

                    btn\_stop.setEnabled(true);

                } catch (Exception exc) {

                    JOptionPane.showMessageDialog(frame, exc.getMessage(),

                            "Error", JOptionPane.ERROR\_MESSAGE);

                }

            }

        });

        btn\_stop.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                if (!isConnected) {

                    return;

                }

                try {

                    boolean flag = closeConnection();

                    if (flag == false) {

                        throw new Exception("Error!");

                    }

                    JOptionPane.showMessageDialog(frame, "Offline!");

                    btn\_start.setEnabled(true);

                    txt\_port.setEnabled(true);

                    txt\_name.setEnabled(true);

                    btn\_stop.setEnabled(false);

                } catch (Exception exc) {

                }

            }

        });

        frame.addWindowListener(new WindowAdapter() {

            public void windowClosing(WindowEvent e) {

                if (isConnected) {

                    closeConnection();

                }

                System.exit(0);

            }

        });

    }

    public boolean connectServer(int port, String hostIp, String name) {

        try {

            socket = new Socket(hostIp, port);

            writer = new PrintWriter(socket.getOutputStream());

            reader = new BufferedReader(new InputStreamReader(socket

                    .getInputStream()));

            sendMessage(name + "#" + socket.getLocalAddress().toString());

            messageThread = new MessageThread(reader, textArea);

            messageThread.start();

            isConnected = true;

            return true;

        } catch (Exception e) {

            textArea.append("Connection failure" + "\r\n");

            isConnected = false;

            return false;

        }

    }

    public void sendMessage(String message) {

        writer.println(message);

        writer.flush();

    }

//close the window by client

    @SuppressWarnings("deprecation")

    public synchronized boolean closeConnection() {

        try {

            sendMessage("CLOSE");

            messageThread.stop();

            if (reader != null) {

                reader.close();

            }

            if (writer != null) {

                writer.close();

            }

            if (socket != null) {

                socket.close();

            }

            isConnected = false;

            return true;

        } catch (IOException e1) {

            e1.printStackTrace();

            isConnected = true;

            return false;

        }

    }

    class MessageThread extends Thread {

        private BufferedReader reader;

        private JTextArea textArea;

        public MessageThread(BufferedReader reader, JTextArea textArea) {

            this.reader = reader;

            this.textArea = textArea;

        }

 //close the thread by sever

        public synchronized void closeCon() throws Exception {

            listModel.removeAllElements();

            if (reader != null) {

                reader.close();

            }

            if (writer != null) {

                writer.close();

            }

            if (socket != null) {

                socket.close();

            }

            isConnected = false;

        }

        public void run() {

            String message = "";

            while (true) {

                try {

                    message = reader.readLine();

                    StringTokenizer stringTokenizer = new StringTokenizer(

                            message, "#");

                    String command = stringTokenizer.nextToken();

                    if (command.equals("CLOSE"))

                    {

                        textArea.append("Sever closed!\r\n");

                        closeCon();

                        return;

                    } else if (command.equals("ADD")) {

                        String username = "";

                        if ((username = stringTokenizer.nextToken()) != null) {

                            listModel.addElement(username);

                        }

                    } else if (command.equals("DELETE")) {

                        String username = stringTokenizer.nextToken();

                        listModel.removeElement(username);

                    } else if (command.equals("USERLIST")) {

                        int size = Integer

                                .parseInt(stringTokenizer.nextToken());

                        String username = null;

                        for (int i = 0; i < size; i++) {

                            username = stringTokenizer.nextToken();

                            listModel.addElement(username);

                        }

                    } else if (command.equals("MAX")) {

                        textArea.append(stringTokenizer.nextToken()

                                + stringTokenizer.nextToken() + "\r\n");

                        closeCon();

                        JOptionPane.showMessageDialog(frame, "Full", "Error",

                                JOptionPane.ERROR\_MESSAGE);

                        return;

                    } else {

                        textArea.append(message + "\r\n");

                    }

                } catch (IOException e) {

                    e.printStackTrace();

                } catch (Exception e) {

                    e.printStackTrace();

                }

            }

        }

    }

}

User.java

package room;

public class User{

    private String name;

    private String ip;

    public User(String name, String ip) {

        this.name = name;

        this.ip = ip;

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public String getIp() {

        return ip;

    }

    public void setIp(String ip) {

        this.ip = ip;

    }

}