A CLIENT-SERVER CHAT PROGRAM

CS 3516 / CS 513 Class Project

Declaration

I confirm that the work for the following report and project are done by myself unless the part marked to the contrary.

Qi Wang ID: 714957460

Jun 29, 2017

Abstract

This report outlines the design and development of a program which achieved the functions of a client -server chat room. The program was written in Java and Windows socket commands to run under the Windows operating system. The program was designed to make maximum use of abstract data types and of software re-use. Particular attention was paid to use multithreading property in Java to let server serves multiple clients at the same time, and run the server and client side under the graphical user interface. In the flowing part of this report, I will give a specific description of the program such as the project description, detailed design, testing and validation of the program and also a conclusion part.

Contents

Project Description

For this project, the main task was to write a chat program based on the client-server model using some language, such as Java and Windows socket commands. There are several general

specifications must be implemented: multiple clients, no graphical user interface is needed for the server, a sample graphical user interface can be implemented for the server, clients must be able to “whisper” to each other, client must be able to choose a nickname. The learning objective of this project is providing us with the insight into practical computer networks and the problems faced when implementing them.

Project Specification

The server:

1. Server operations (such as connect requests and disconnect requests) should be printed out by the server.
2. The server must handle connections / disconnections without disruption of other services.
3. Clients must have unique nicknames, duplicates must be resolved before allowing a client to be connected.
4. All clients must be informed of changes in the list of connected users.

The client:

1. A list of online users must be displayed (via GUI or command).
2. Connection / disconnection actions of users must be displayed.
3. Messages from the originating user and other users must be displayed (in other words the messages you send must also be displayed).
4. Must still be able to receive messages / actions while typing a message.
5. Clients must be able to disconnect without disrupting the server.

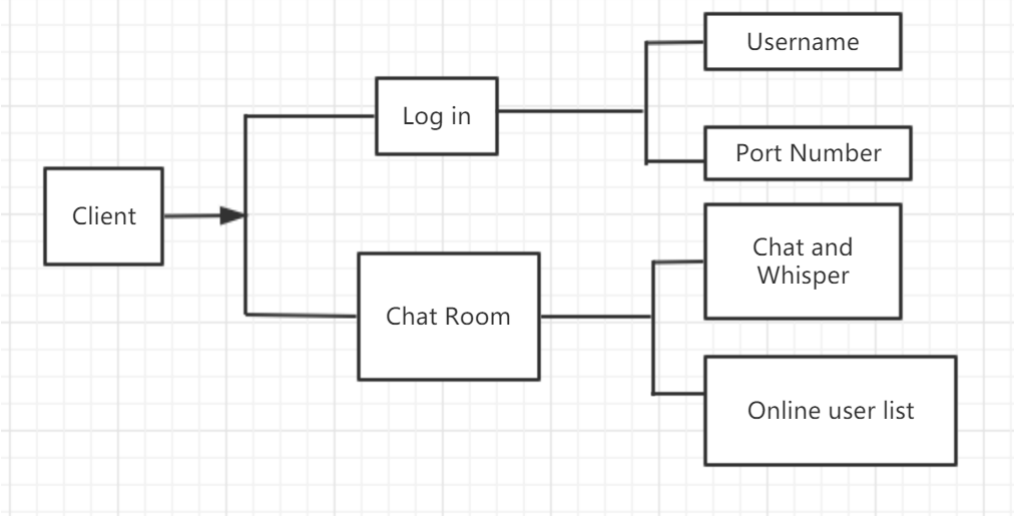
The requirements for server and client side are simply shown in the structure chart below.

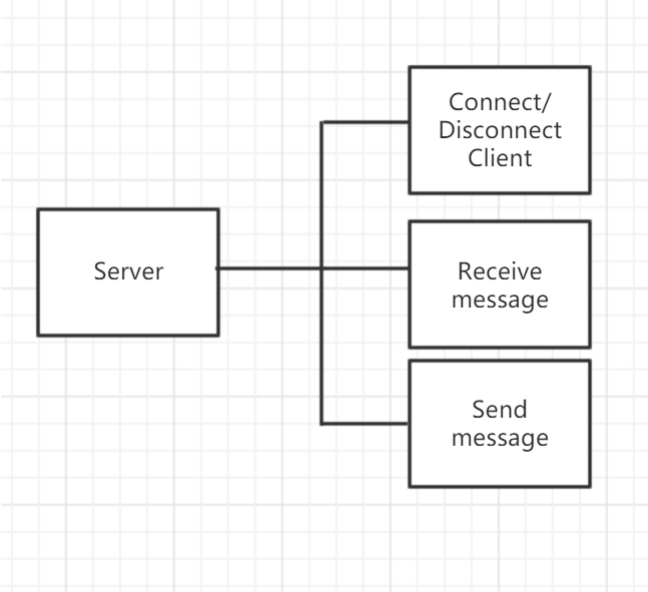
Detailed Design

The system was design to be C/S structure design pattern. In this way, the server will receive the message sent by client, and then the server will report the message to other clients. The client side has two graphical user interface to display all the functions that the client can use. The first interface achieves the log in function. The second interface will appear after logging in and the client will chat through this interface. The server side has a sample graphical user interface.

There are totally three packages of the program, Server, Client and Util. Server package includes the code to build the server side, send and receive messages. The client package includes the code to achieve the log in and chat room user interface, as well as the establish of client thread. The Util package processes the XML documents and messages.

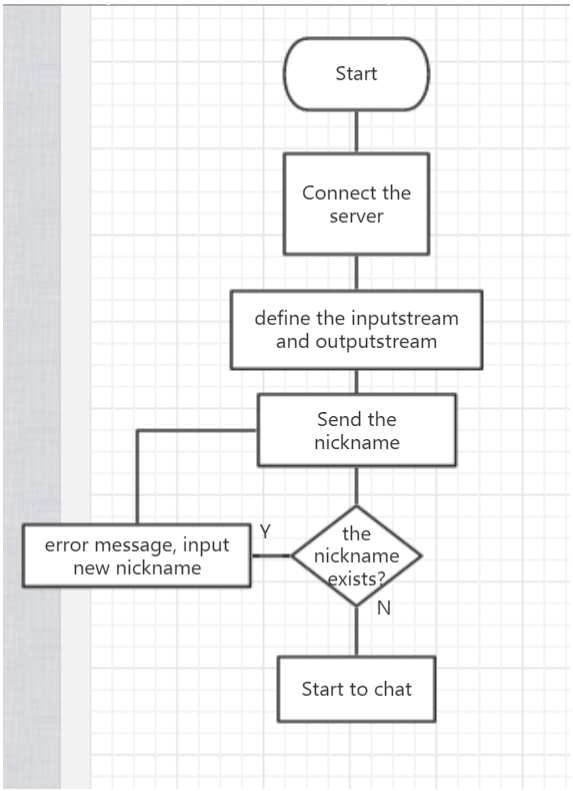
The figures below simply show the structure of the program.

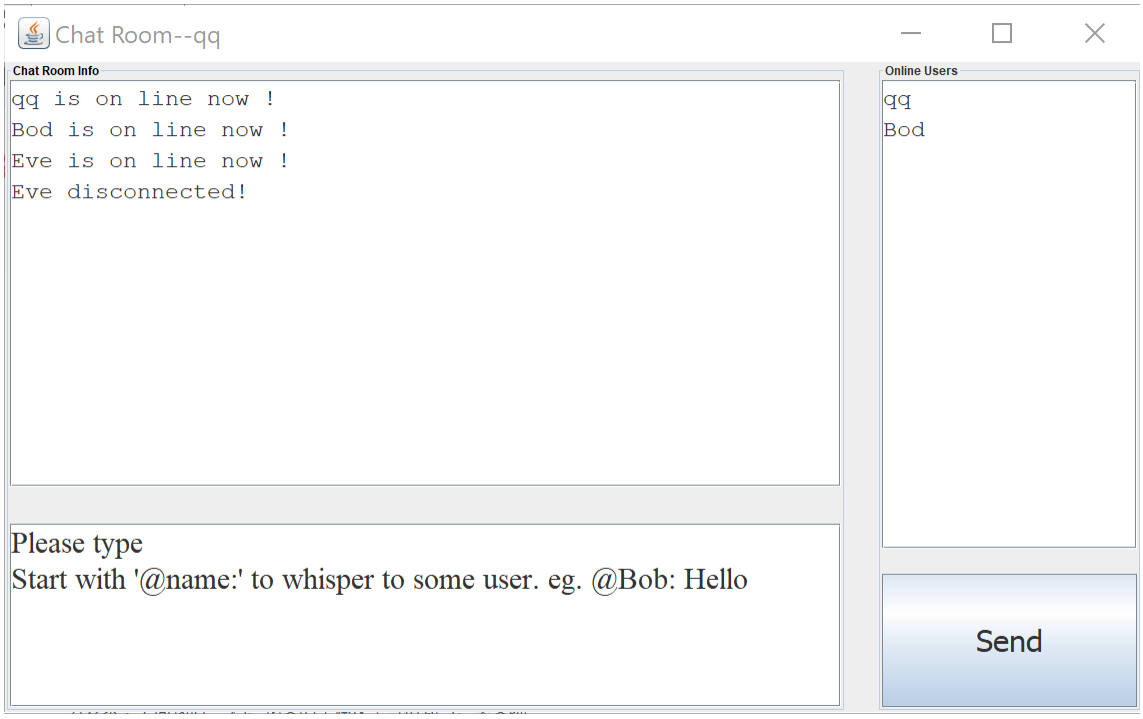




User Log in

The user log in interface can make the connection between the client and server. If successful, the client can start to chat. Since the requirements said every client must have a unique nickname. So, we have to check the nickname first. The flowchart shows the process of user log in.



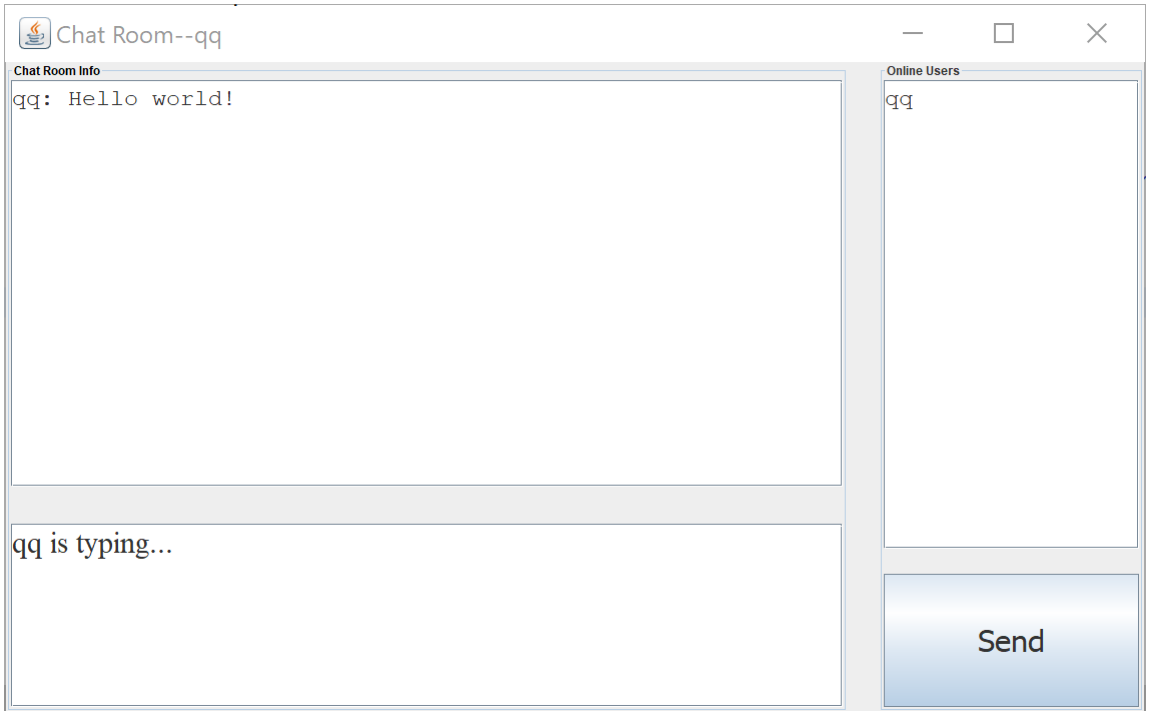


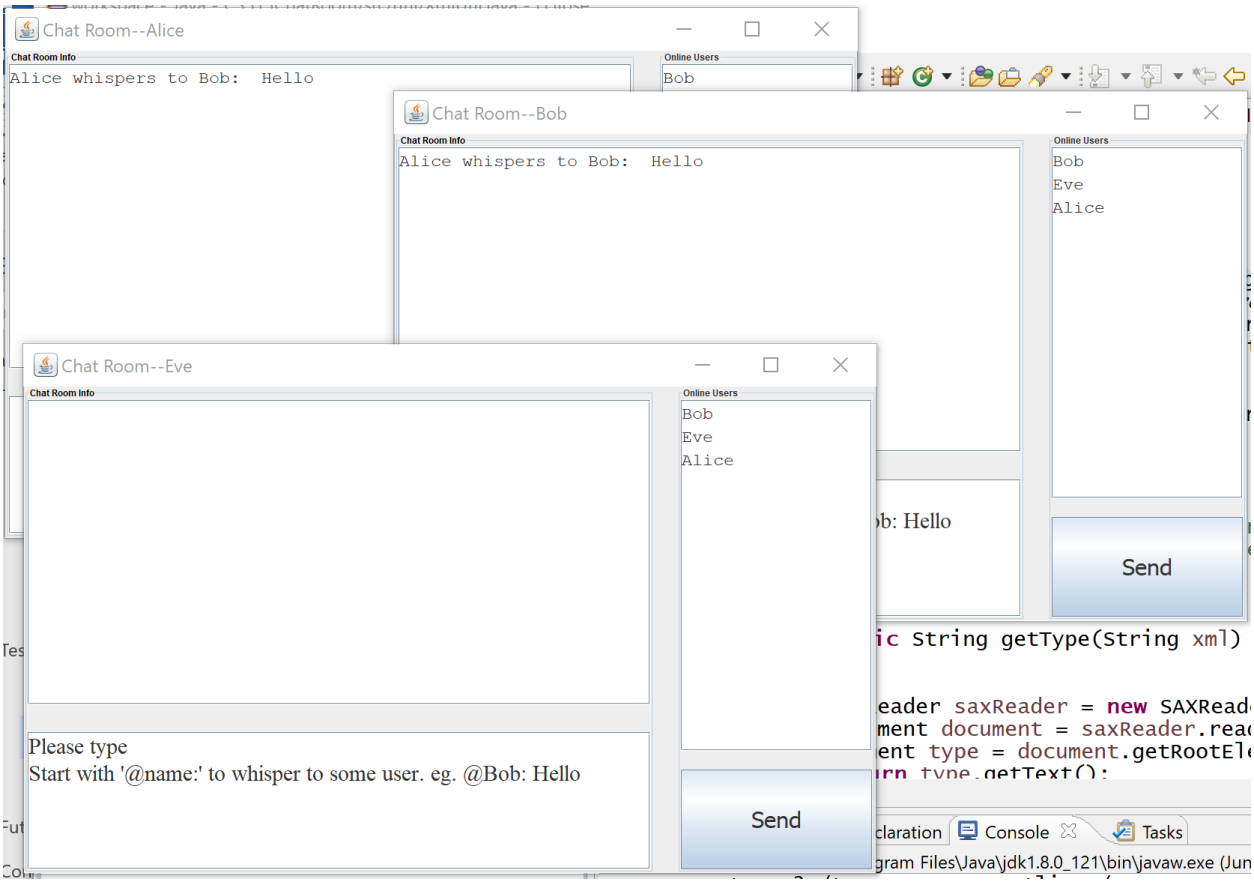
Chat and whisper

After logging in, the client will enter the chat room interface, with his or her name shown in the top of the left side, as is shown in the figure. The client qq has entered the chat room. Then she

can type the message that she wants to send to others. The message will be first send to the server side, and the server will report that message to multiple client. As a result, the message will be shown in the chat interface.

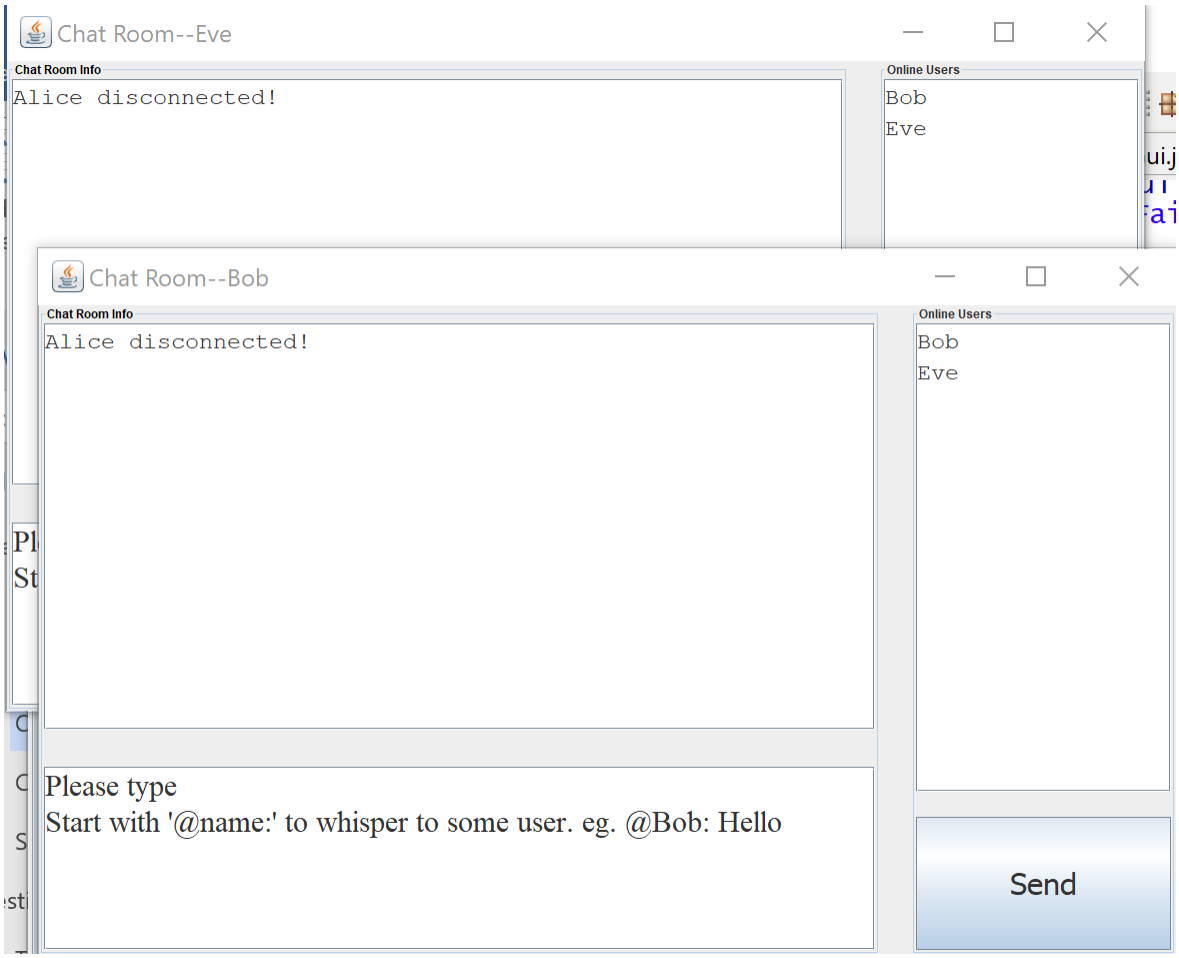
If the client wants to whisper to some user, she or he needs to add "@nickname:" prefix before the message. The meaning is that she or he whispers to the person that has the nickname. For example, as is shown in the figure, there are three clients, Alice, Bob and Eve. Alice typed "@Bob: hello". Then only Bob will receive the message.





Online user list

The online user list will be shown in the right side of the chat room interface. This list of online users will be updated every time when some client has connected to the server or disconnected to the server. Look at the former example, now the three clients are shown in the user list. Then if Alice chooses to leave, the interface of the rest clients only shows Bob and Eve.

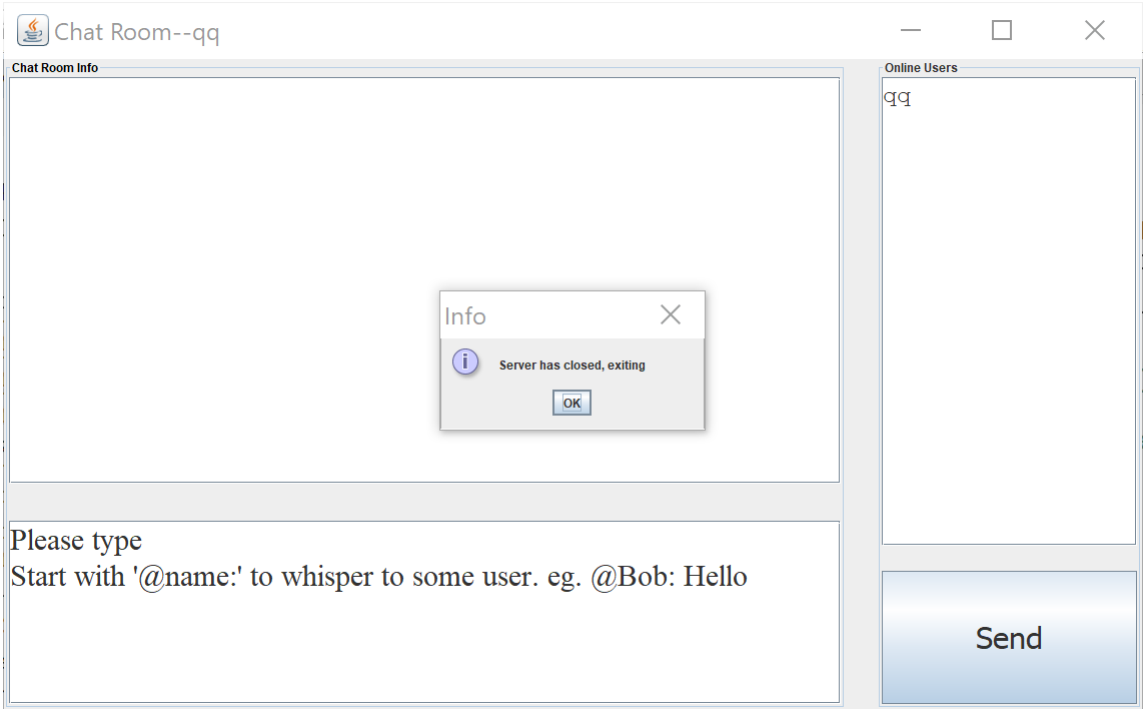


Clients disconnect

The client can choose to disconnect by clicking the close button of the chat room interface. This operation will send a message to the server side. Then the server will update the online user list and tell other clients that the user has left. The figure in last part can show the details.

Server disconnects

If the Server disconnects, the server socket will be closed and the server can not server for the clients. When this happened, there will be a new window to tell the clients the server can not work for them. The whole system will stop after the server close the new window. As the figure shows below.



Function realization

This program uses the XML document and the dom4j to help to build and read the XML document. Using XML provides a lot of convenience. The reason lies in the fact that it has labels that makes it easy to understand the type of the message, such as whether it is a user log in message or a chat message.

If the server or client wants to send a message, the message will be written as a XML document first, then send to the receiver. When receiving the message, the message will be read according to its labels.

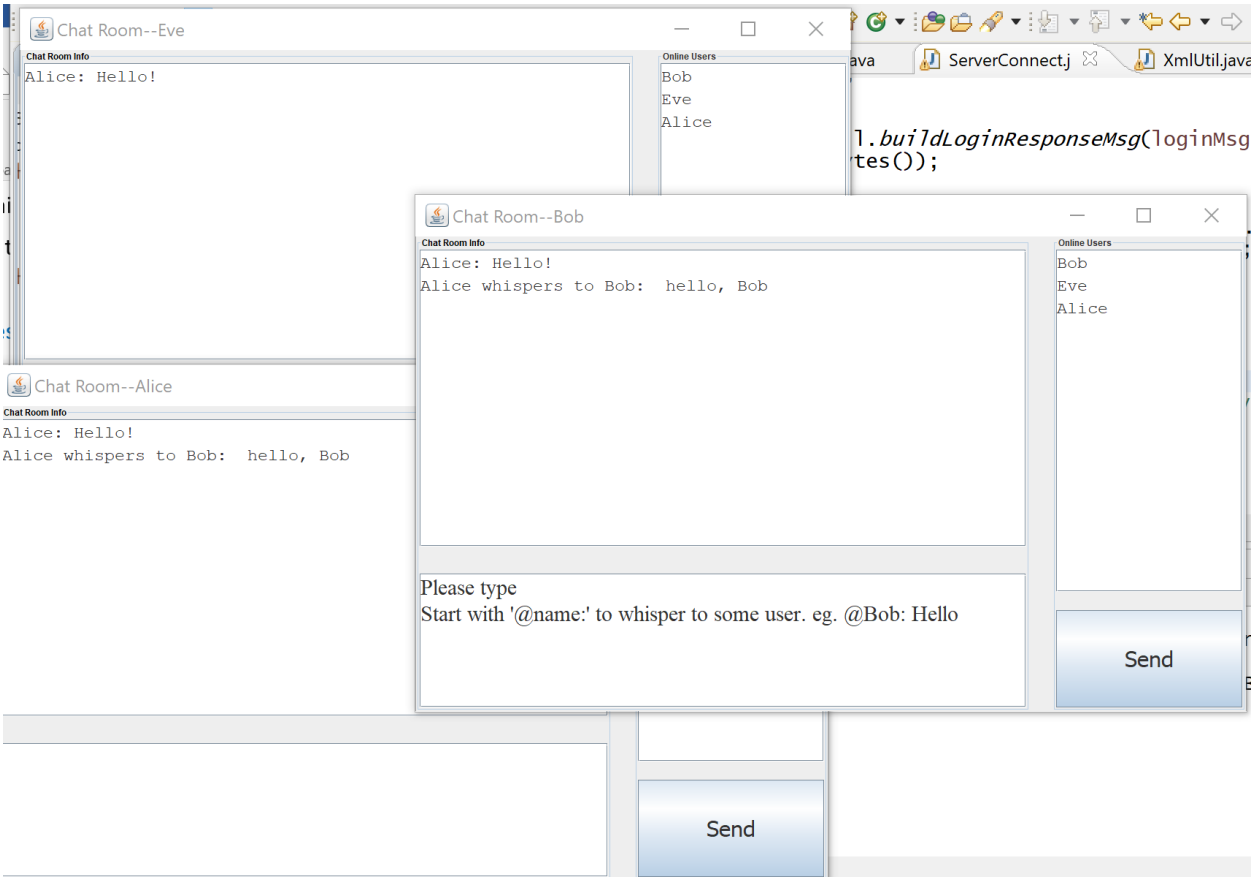
Testing and Validation

In the following part, I will do some testing to make sure that the chat program works well.

Testing for basic function

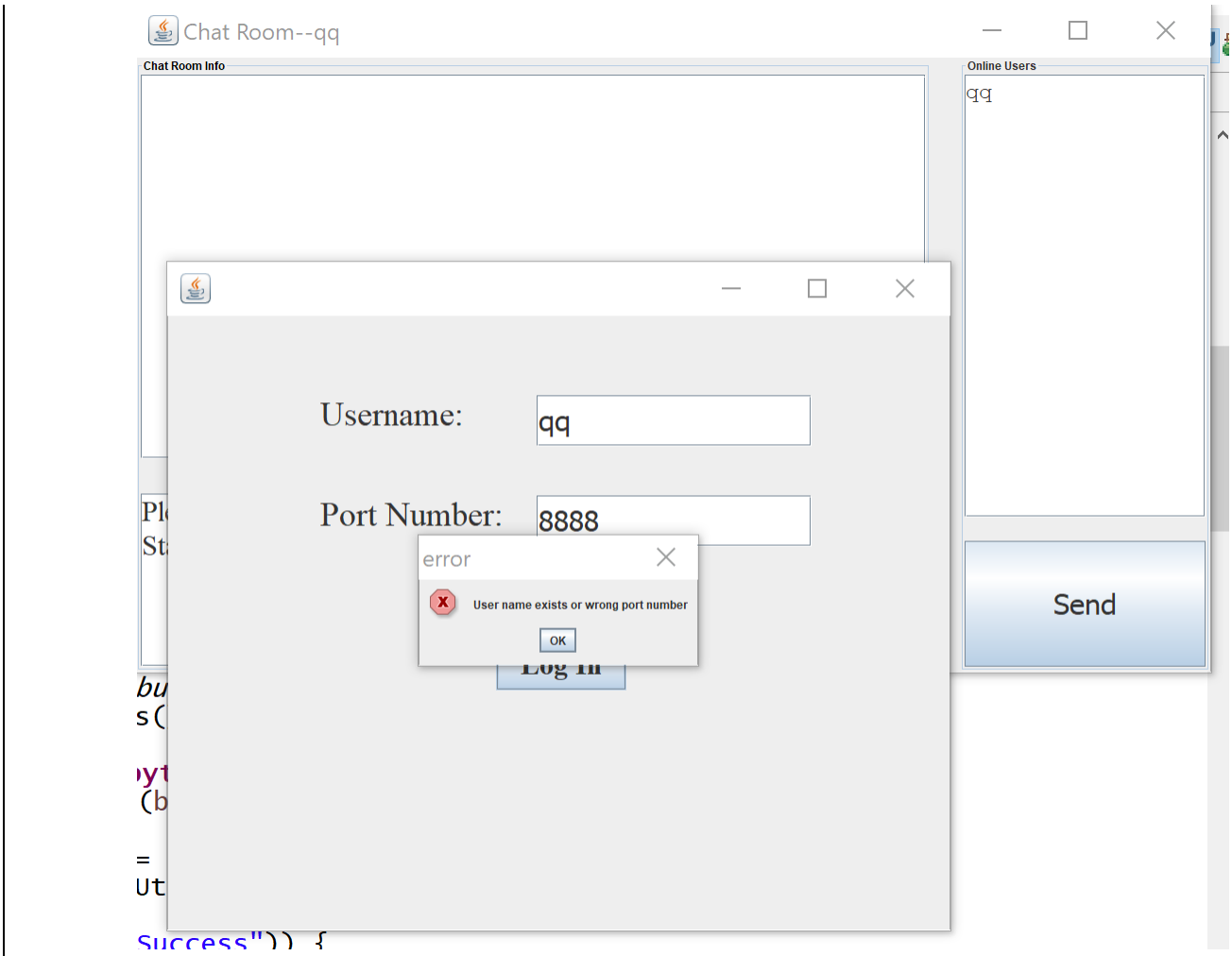
To run the program, first run the ServerGui.java file and press start button to start the server. And then run ClientGui.java to enter username and log in. After that the user interface for chatting will appear. Th client can type their message in the text area and click Send button to send message to all people. As is shown in the figure below, Alice said hello to all the people.

To test the whisper function, type “@Bob:” before the message, then the message will be only sent to Bob.



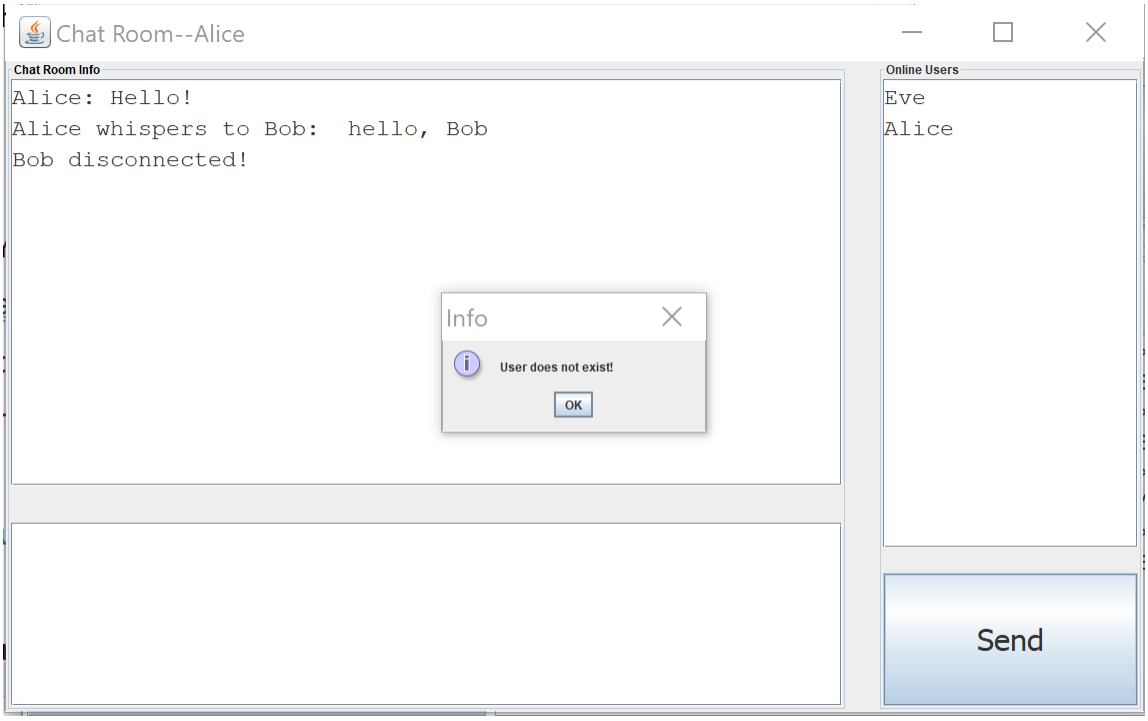
The duplicate nickname when logging in

As the project required, the nickname of clients are unique. Therefore, when logging in, the server will check whether the username has been used by other clients first. If the username has been used by other clients, the new client must change a new name. For example, if the client qq is online, another client can not use the same name qq to log in. And there will be an error message to inform the client to do that.



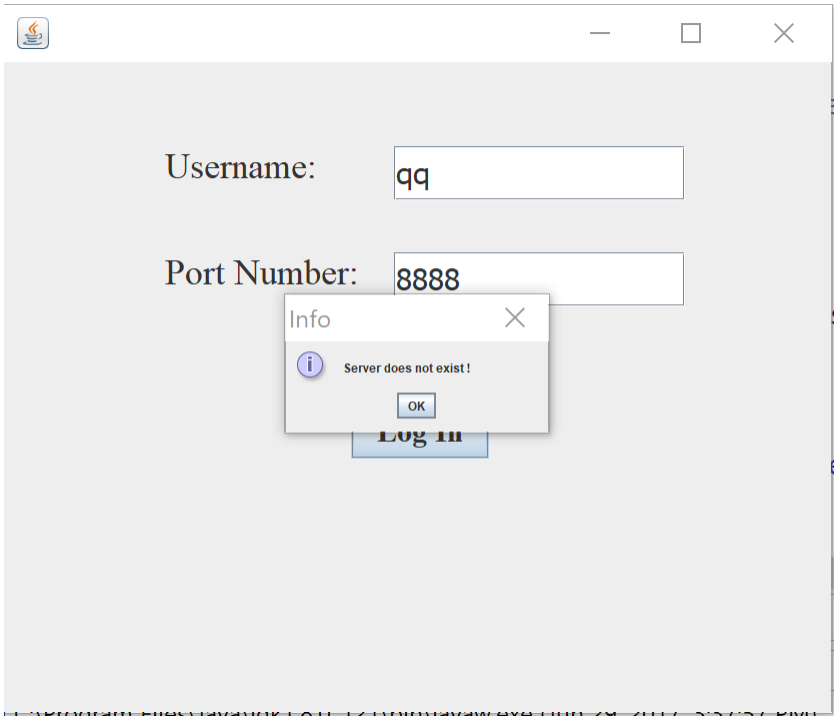
Whispers to a non-existing user

Now let Bob disconnect the server to see if the system is stable if Alice whispers to a non-existing client. After sending the "@Bob: hi Bob" message, the system tells Alice that Bob is not online and She can type another message.



Connect to Non-Existing Server

Test if the system can stay stable when a client is trying to connect to a non-existing server. So, run the ClientGui.java file and press log in button without starting the accordingly server. The system will notice the clients this problem and display an error message to let the user try a different server.



Future Development

There is always a room for improvements in any software package. So there a few things should be improved in the future. First of all, in the chat room user interface, words in the info box have the same front although the message were sent from different source, the server, or the clients. Different fronts or colors can make it easy to tell whether it is a message from server or clients. Next, right now the chat room program only supports text transfer. However, as a real chat room, it should be able to send files of different formats. In addition, other functions such as voice chat will enhance the program to a higher level and make is more useful in the modern world.

Conclusion – Solution Summary

This report has described the successful design and development of a client-server chat room program using java and Windows socket. Use multithreading in server and client side to make sure that a server can work well when there are multiple clients. The different clients can send and receive messages without influencing each other. Besides, the graphical user interfaces are made using Java. For transferring message between server and clients, using inputstream and outputstream to transfer XML message can make a high efficiency work.

Appendices

Source code

```
package server;

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

import util.XmlUtil;

import java.util.*;
import java.awt.Window.Type;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
/**
 * Creates the start server graphical user interface. After click the
start button the server
 * thread will be started and wait for any client to connect to this
server.
 * @author QiWang
 *
 */
public class ServerGui extends JFrame{

	private JPanel jPanel1;
	private JLabel jLabel2;
	private JLabel jLabel1;
	private JButton jButton1;
	private JTextField jTextField1;

	private Map<String, ServerChannel> map;
```

```
	public ServerGui() {

		map = new HashMap<String, ServerChannel>();
		setLayout();
	}
	/**
	 * Sets the locations of the user interface. Specifically, define
the function of satrt button
	 * and close window button.
	 */
	public void setLayout() {
		jLabel2 = new JLabel();
		jLabel2.setFont(new Font("Tahoma", Font.PLAIN, 22));
		jLabel2.setSize(190, 25);
		jLabel2.setLocation(225, 200);
		jPanel1 = new JPanel();
		jLabel1 = new JLabel();
		jLabel1.setBounds(21, 64, 202, 42);
		jLabel1.setFont(new Font("Times New Roman", Font.PLAIN,
36));
		jTextField1 = new JTextField(15);
		jTextField1.setBounds(233, 64, 325, 43);
		jTextField1.setFont(new Font("Tahoma", Font.PLAIN, 30));
		jButton1 = new JButton();
		jButton1.setFont(new Font("Times New Roman", Font.PLAIN,
30));
		jButton1.setBounds(239, 137, 182, 35);

		jPanel1.setName("Server");
		this.setName("Start Server");
		this.setType(Type.POPUP);
		this.setTitle("Start Server");
		jLabel1.setText("Port Number: ");
		jTextField1.setText("8888");
		jButton1.setText("Start Server");
		jPanel1.setLayout(null);

		jPanel1.add(jLabel1);
		jPanel1.add(jTextField1);
		jPanel1.add(jButton1);
		jPanel1.add(jLabel2);

		this.getContentPane().add(jPanel1);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setAlwaysOnTop(true);
		this.setVisible(true);
		this.pack();

		Toolkit tk = Toolkit.getDefaultToolkit();
		Dimension dim = tk.getScreenSize();
		int width = dim.width / 4;
		int height = dim.height / 3;
		this.setSize(width, height);

		int xPos = (dim.width / 2) - (this.getWidth() / 2);
```

```
            int yPos = (dim.height / 2) - (this.getHeight() / 2);
            this.setLocation(xPos, yPos);

            ListenForButton lForButton = new ListenForButton();
            jButton1.addActionListener(lForButton);

            ListenForWindow lForWindow = new ListenForWindow();
            this.addWindowListener(lForWindow);

      }
      /**
       * If the server has been closed, send the message to all clients
and close the system.
       * @author QiWang
       *
       */
      private class ListenForWindow extends WindowAdapter {


            public void windowClosing(WindowEvent e) {
                  try {
                        String xml =
XmlUtil.buildCloseServerWindowMsg();
                        for(ServerChannel channel:
ServerGui.this.getMap().values()) {
                              channel.send(xml);
                        }
                  } catch (Exception e1) {
                        // TODO Auto-generated catch block
                        e1.printStackTrace();
                  } finally {
                        System.exit(0);
                  }

            }


      }
      /**
       * Strat the server thread and wait for any client to connect to
the server.
       * @author QiWang
       *
       */
      private class ListenForButton implements ActionListener{

            @Override
            public void actionPerformed(ActionEvent e) {
                  ServerGui.this.execute(e);
            }

      }

      public void execute(ActionEvent evt) {
            int port
=Integer.parseInt(this.getjTextField1().getText()) ;
            new Thread( new ServerConnect(this, port)).start();
      }
```

```
public Map<String, ServerChannel> getMap() {
	return map;
}

public void setMap(Map<String, ServerChannel> map) {
	this.map = map;
}


public void setjPanel1(JPanel jPanel1) {
	this.jPanel1 = jPanel1;
}


public JLabel getjLabel1() {
	return jLabel1;
}



public void setjLabel1(JLabel jLabel1) {
	this.jLabel1 = jLabel1;
}


public JButton getjButton1() {
	return jButton1;
}


public void setjButton1(JButton jButton1) {
	this.jButton1 = jButton1;
}


public JTextField getjTextField1() {
	return jTextField1;
}


public void setjTextField1(JTextField jTextField1) {
	this.jTextField1 = jTextField1;
}



public JLabel getjLabel2() {
	return jLabel2;
}

public void setjLabel2(JLabel jLabel2) {
	this.jLabel2 = jLabel2;
}

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

```
}
```

```
package server;

import java.io.IOException;
import java.net.*;
import java.io.*;
import util.XmlUtil;

import javax.swing.JOptionPane;
/**
 * Implements the server thread concerning to log in process. Meanly
has two steps, first start
 * the server to wait for any client to connect to this server.
Second, start the thread if any
 * client has connect to this server, check if the client can enter
the chat room.
 *
 * @author QiWang
 *
 */
public class ServerConnect implements Runnable{

	private ServerSocket server;
	private ServerGui serverGui;
	private boolean isRunning = true;
	public static final int BUFF_LENGTH = 5000;

/**
 *
 * @param serverGui   Get the log in user interface in order to send
error message in case.
 * @param portNumber  Get the port number to start the server.
 */
	public ServerConnect(ServerGui serverGui, int portNumber) {
		super();
		try {
			this.serverGui = serverGui;
			this.server = new ServerSocket(portNumber);
			this.serverGui.getjLabel2().setText("Server is
working");
		} catch (IOException e) {
			e.printStackTrace();
			JOptionPane.showMessageDialog(this.serverGui, "Invalid
port number!", "Warning", JOptionPane.ERROR_MESSAGE);
		}
	}


/**
 * Check if the user name has been used by other clients and return
the success or failure result
 * back to the client. If success, start a new thread for the client
and update the online user list.
 */
	@Override
```

```
	public void run() {
		while(this.isRunning ) {
			try {
				Socket client  = this.server.accept();

				InputStream dis = client.getInputStream();
				OutputStream dos = client.getOutputStream();

				byte[] buffer = new byte[BUFF_LENGTH];
				int length = dis.read(buffer);

				String loginXml = new String(buffer, 0, length);
				String userName = XmlUtil.getUserName(loginXml);
				String loginMsg;
				boolean isLogin = false;

				if
(this.serverGui.getMap().containsKey(userName)) {
					loginMsg = "Fail";
				} else {
					loginMsg = "Success";
					isLogin = true;
				}

				String xml =
XmlUtil.buildLoginResponseMsg(loginMsg);
				dos.write(xml.getBytes());

				if (isLogin) {
					ServerChannel channel = new
ServerChannel(this.serverGui, client);
					this.serverGui.getMap().put(userName,
channel);
					channel.updateUserList(userName);

					new Thread(channel).start();
					String str =userName +  " is online now!";
					String msg = XmlUtil.buildServerMsg(str);

				}

			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

}


package server;

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

import util.CloseUtil;
import util.XmlUtil;
```

```java
/**
 * A class used to transfer message between client and server. The
server can send  and receive
 * message through this class. To see if the client has sent chat
message of a client has disconnected.
 * @author QiWang
 *
 */
public class ServerChannel implements Runnable{

	private static final int BUFF_LENGTH = 5000;
	private ServerGui serverGui;
	private InputStream dis;
	private OutputStream dos;
	private boolean isRunning = true;

	public ServerChannel(ServerGui serverGui,  Socket client) {
		try {
				this.serverGui = serverGui;
				this.dos = client.getOutputStream();
				this.dis = client.getInputStream();
		} catch (IOException e) {
			isRunning = false;
			CloseUtil.closeAll(dis, dos);
		}
	}
	/**
	 * Used to send message through input and output stream.
	 * @param msg
	 */
	public void send(String msg) {

		try {
				dos.write(msg.getBytes());
				dos.flush();
		} catch (IOException e) {
			e.printStackTrace();
			this.isRunning = false;
			CloseUtil.closeAll(dos);
		}
	}

	public String updateUserList(String name) {
		Set<String> users = this.serverGui.getMap().keySet();
		String xml = XmlUtil.buildUserList(users);
		for (ServerChannel channel:
this.serverGui.getMap().values()) {
			channel.send(xml);
		}
		if(name != null) {
			return name;
		}
		return null;
	}

/**
 * Processes the chat and whisper message.
```

```
 * If a client has left, update the user list and inform other
clients.
 */
      @Override
      public void run() {
            while(isRunning) {

                  try {
                        byte[] buffer = new byte[BUFF_LENGTH];
                        int length = this.dis.read(buffer);
                        String xml = new String(buffer, 0, length);
                        System.out.println(xml.toString());
                        int type =
Integer.parseInt(XmlUtil.getType(xml));

                        switch(type) {
                        /** client sent message */
                              case 2:
                                    String userName =
XmlUtil.getUserName(xml);
                                    String content = XmlUtil.getMsg(xml);
                                    String message = userName + ": " +
content;
                                    String msgXml =
XmlUtil.buildServerMsg(message);
                                    Map<String, ServerChannel> map =
this.serverGui.getMap();
                                    if (content.startsWith("@") &&
content.indexOf(":") > -1) {

                                          String name =
content.substring(1, content.indexOf(":"));
                                          String c =
content.substring(content.indexOf(":") + 1);
                                          String str = userName + "
whispers to " + name + ": " + c;
                                          String msg =
XmlUtil.buildServerMsg(str);

                                          if (!map.containsKey(name)) {
                                                String error =
XmlUtil.buildNonUserResponse();

                                                this.send(error);
                                                break;
                                          }
                                          this.send(msg);
                                          for (String s: map.keySet()) {
                                                if (s.equals(name)) {
                                                      map.get(s).send(msg);

                                                }
                                          }
                                    } else {
                                          for(ServerChannel channel:
map.values()) {
                                                channel.send(msgXml);
                                          }
                                    }
```

```
                                    break;
                                /** client closed window */
                                case 3:
                                    String userName1 =
XmlUtil.getUserName(xml);
                                    ServerChannel channel =
this.serverGui.getMap().get(userName1);
                                    String response =
XmlUtil.buildCloseClientWindowResponse();
                                    channel.send(response);

      this.serverGui.getMap().remove(userName1);
                                    this.updateUserList(null);
                                    this.dis.close();
                                    this.dos.close();

                                    String str = userName1 + "
disconnected!";
                                    String msg =
XmlUtil.buildServerMsg(str);
                                    for(ServerChannel c:
this.serverGui.getMap().values()) {
                                        c.send(msg);
                                    }
                                    break;
                                /** Send the message that a new client is
on line */
                                case 10:
                                    String userName2 =
XmlUtil.getUserName(xml);
                                    String str1 = userName2 + " is on line
now !";
                                    String msg1 =
XmlUtil.buildServerMsg(str1);
                                    for(ServerChannel c:
this.serverGui.getMap().values()) {
                                        c.send(msg1);
                                    }
                            }

                        } catch (IOException e) {

                        }
                    }
            }


}


package client;

import java.awt.Dimension;
import java.awt.HeadlessException;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.*;
import java.awt.Font;
/**
 * Creates the server log in user interface. Sets the locations of
different elements in the
 * interface. Sends the nickname and port number to the server side
and gets response.
 * If success, creates the chat room user interface and starts the
client thread.
 * @author QiWang
 *
 */
public class ClientGui extends JFrame{

	private JPanel jPanel1;
	private JButton jButton1;
	private JLabel jLabel1;
	private JLabel jLabel12;
	private JTextField username;
	private JTextField portNumber;



	public ClientGui() throws HeadlessException {
		super();
		this.setLayout();
	}

	public void setLayout() {
		this.jPanel1 = new JPanel();

		this.jLabel1 = new JLabel();
		jLabel1.setFont(new Font("Times New Roman", Font.PLAIN,
36));
		jLabel1.setBounds(161, 84, 177, 41);
		jLabel1.setText("Username: ");

		this.jLabel12 = new JLabel();
		jLabel12.setFont(new Font("Times New Roman", Font.PLAIN,
36));
		jLabel12.setBounds(161, 190, 222, 41);
		jLabel12.setText("Port Number: ");

		this.jButton1 = new JButton();
		jButton1.setFont(new Font("Times New Roman", Font.BOLD,
30));
		jButton1.setBounds(347, 342, 138, 54);
		jButton1.setText("Log In");

		this.username = new JTextField();
		username.setFont(new Font("Tahoma", Font.PLAIN, 30));
		username.setBounds(390, 84, 291, 54);
		username.setText("qq");

		this.portNumber = new JTextField();
		portNumber.setFont(new Font("Tahoma", Font.PLAIN, 30));
		portNumber.setBounds(390, 190, 291, 54);
```

```
            portNumber.setText("8888");
            jPanel1.setLayout(null);

            jPanel1.add(jLabel1);
            jPanel1.add(jLabel12);
            jPanel1.add(jButton1);
            jPanel1.add(username);
            jPanel1.add(portNumber);
            this.getContentPane().add(jPanel1);

            this.setVisible(true);

            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension dim = tk.getScreenSize();
            int width = dim.width / 3;
            int height = dim.height / 2;

            this.setSize(width, height);

            int xPos = (dim.width / 2) - (this.getWidth() / 2);
            int yPos = (dim.height / 2) - (this.getHeight() / 2);

            this.setLocation(xPos, yPos);

            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            ListenForButton lForButton = new ListenForButton();
            this.jButton1.addActionListener(lForButton);

      }

      private class ListenForButton implements ActionListener {

            @Override
            public void actionPerformed(ActionEvent e) {
                  ClientGui.this.logIn(e);
            }

      }
      /**
       * Sends the log in information to the server and get result from
server.
       * Creates new client thread or displays error message.
       * @param evt The event of pressing button.
       */
      private void logIn(ActionEvent evt) {
            String username = this.username.getText();
            String portNumber = this.portNumber.getText();
            ClientConnect clientConnect = new ClientConnect(this,
Integer.parseInt(portNumber), username);

            if (clientConnect.logIn()) {
                  new Thread(clientConnect).start();
            } else {
                  JOptionPane.showMessageDialog(this, "User name exists
or wrong port number", "error",
                              JOptionPane.ERROR_MESSAGE);
            }
```

```
	}


	public static void main(String[] args) {
		new ClientGui();
	}

}


package client;

import java.net.*;
import java.util.List;

import javax.swing.JOptionPane;

import util.XmlUtil;

import java.io.*;
/**
 * This class creates the clients thread to make connection with the
server.
 * It can build messages and send it to server. And also can read the
message from the server.
 * @author QiWang
 *
 */
public class ClientConnect implements Runnable{

	private static final int BUFF_LENGTH = 5000;
	private String hostAddress = "localhost";
	private int portNumber;
	private String username;
	private ClientGui clientGui;
	private Socket client;
	private InputStream dis;
	private OutputStream dos;
	private ClientChat clientChat;

	private boolean isRunning = true;
	/**
	 * Create a new server socket.
	 * @param clientGui The log in interface.
	 * @param portNumber Get port number from the log in user
interface.
	 * @param username Get the user name from the log in user
interface.
	 */
	public ClientConnect(ClientGui clientGui, int portNumber, String
username) {
		this.clientGui = clientGui;
		this.portNumber = portNumber;
		this.username = username;
		this.connect();
	}
```

```
	public Socket getClient() {
		return client;
	}



	public String getUsername() {
		return username;
	}



	private void connect() {
		try {
			this.client = new Socket(this.hostAddress,
this.portNumber);
			this.dis = this.client.getInputStream();
			this.dos = this.client.getOutputStream();

		} catch (IOException e) {
			JOptionPane.showMessageDialog(clientGui, "Server does
not exist !", "Info", JOptionPane.INFORMATION_MESSAGE);

		}
	}

	public boolean logIn() {

		try {
			String xml = XmlUtil.buildLoginMsg(this.username);
			dos.write(xml.getBytes());

			byte[] buffer = new byte[BUFF_LENGTH];
			int length = dis.read(buffer);

			String loginResponse = new String(buffer, 0, length);
			String response =
XmlUtil.getLoginResponse(loginResponse);

			if (response.equals("Success")) {
				this.clientChat = new ClientChat(this);
				this.clientGui.setVisible(false);
				String msg =
XmlUtil.buildAddUser(this.username);
				dos.write(msg.getBytes());
				return true;
			}


		} catch (IOException e) {

		}
		return false;
	}

	public void sendMsg(String message, int type) {
```

```
            try {
                  int num = type;
                  String xml = "";
                  if (type == 2) {
                        xml = XmlUtil.buildMsg(this.username, message);
                  } else if (type == 3) {
                        xml =
XmlUtil.buildCloseClientWindowMsg(this.username);
                  }
                  this.dos.write(xml.getBytes());
            } catch (IOException e) {

                  e.printStackTrace();
            }
      }
/**
 * Reads the message sent by server, and gives the response according
to different kinds of messages.
 */
      @Override
      public void run() {
            try {
                  while(isRunning) {
                        byte[] buffer = new byte[BUFF_LENGTH];
                        int length = this.dis.read(buffer);

                        String xml = new String(buffer, 0, length);
                        int type =
Integer.parseInt(XmlUtil.getType(xml));
                        /**server message*/
                        if (type == 4) {
                              String content = XmlUtil.getMsg(xml);

      this.clientChat.getjTextArea1().append(content + "\n");
                        /**server window has closed*/
                        } else if (type == 5) {
                              JOptionPane.showMessageDialog(clientChat,
"Server has closed, exiting", "Info",
JOptionPane.INFORMATION_MESSAGE);
                              System.exit(0);
                        /**client closed window response*/
                        }else if (type == 7) {
                              try {

      this.getClient().getInputStream().close();

      this.getClient().getOutputStream().close();
                                    this.getClient().close();
                              } catch (Exception e) {
                              } finally {
                                    System.exit(0);
                              }
                        /** update the online user list*/
                        } else if (type == 8) {
                              String str = "";
                              List<String> names =
XmlUtil.getUserList(xml);
                              for(String name: names) {
```

```
                                str = str + name + "\n";

                            }

        this.clientChat.getjTextArea2().setText(str);
                        /** handle the situation if the client sends
message to a non-existing client*/
                        } else if (type == 9) {
                                JOptionPane.showMessageDialog(clientChat,
"User does not exist!", "Info", JOptionPane.INFORMATION_MESSAGE);
                        }

                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }

        }

}


package client;

import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusAdapter;
import java.awt.event.FocusEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.*;


import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import java.awt.Insets;
import java.awt.Color;
import java.awt.Font;
import javax.swing.border.TitledBorder;

import server.ServerChannel;
import server.ServerGui;

import util.XmlUtil;
/**
 * Creates the chat room graphical user interface. Sets the locations
of elements in the user interface.
 * Objects of this class will be created when the client has
successfully logged in. Sends messages
 * to the server if the client has sent chat messages to other client
or disconnected the server.
 * @author QiWang
 *
```

```
 */
public class ClientChat extends JFrame{

	private ClientConnect clientConnect;

	private JPanel jPanel1;
	private JPanel jPanel2;
	private JPanel jPanel3;
	private JScrollPane jScrollPane1;
	private JScrollPane jScrollPane2;
	private JScrollPane jScrollPane3;

	private JButton jButton1;

	private JTextArea jTextArea1;
	public JTextArea getjTextArea1() {
		return jTextArea1;
	}

	public void setjTextArea1(JTextArea jTextArea1) {
		this.jTextArea1 = jTextArea1;
	}

	public JTextArea getjTextArea2() {
		return jTextArea2;
	}

	public void setjTextArea2(JTextArea jTextArea2) {
		this.jTextArea2 = jTextArea2;
	}

	private JTextArea jTextArea2;
	private JTextField jTextField1;
	private JTextArea jTextArea3;

	public ClientChat(ClientConnect clientConnect ) {
		super();
		this.clientConnect = clientConnect;
		this.setLayout();

	}


	public void setLayout() {
		this.jPanel1 = new JPanel();
		this.jPanel2 = new JPanel();
		this.jPanel3 = new JPanel();

		this.jScrollPane1  = new JScrollPane();
		this.jScrollPane2 = new JScrollPane();
		this.jScrollPane3 = new JScrollPane();

		this.jButton1 = new JButton();
		jButton1.setFont(new Font("Tahoma", Font.PLAIN, 30));

		this.jTextArea1 = new JTextArea();
		jTextArea1.setFont(new Font("Monospaced", Font.PLAIN, 23));
		this.jTextArea2 = new JTextArea();
```

```
            jTextArea2.setFont(new Font("Monospaced", Font.PLAIN, 23));
            this.jTextField1 = new JTextField();

            this.jTextArea3 = new JTextArea();
            jTextArea3.setFont(new Font("Times New Roman", Font.PLAIN,
30));

            ListenForFocus l = new ListenForFocus();
            jTextArea3.setText("Please type \nStart with '@name:' to
whisper to some user. eg. @Bob: Hello");
            jTextArea3.addFocusListener(l);

            this.jScrollPane1.setViewportView(jTextArea1);
            this.jScrollPane2.setViewportView(jTextArea2);
            this.jScrollPane3.setViewportView(jTextArea3);
            jPanel1.setLayout(new BorderLayout(18, 17));
            this.jPanel1.add(jScrollPane1, BorderLayout.NORTH);

            this.jPanel1.add(jPanel3);

            BorderLayout bl_jPanel2 = new BorderLayout();
            bl_jPanel2.setVgap(25);
            bl_jPanel2.setHgap(18);
            jPanel2.setLayout(bl_jPanel2);
            this.jPanel2.add(jScrollPane2, BorderLayout.NORTH);
            this.jPanel2.add(jButton1);

            this.jPanel3.setLayout(new BorderLayout());
            this.jPanel3.add(jScrollPane3, BorderLayout.SOUTH);

            getContentPane().setLayout(new BorderLayout());
            this.getContentPane().add(jPanel1, BorderLayout.WEST);
            this.getContentPane().add(jPanel2, BorderLayout.EAST);

            jTextArea1.setEditable(false);
            jTextArea1.setColumns(36);
            jTextArea1.setRows(13);

            jTextArea2.setEditable(false);
            jTextArea2.setColumns(18);
            jTextArea2.setRows(15);


            jTextArea3.setColumns(36);
            jTextArea3.setRows(5);
            jTextArea3.setLineWrap(true);

            jPanel1.setBorder(new
TitledBorder(UIManager.getBorder("TitledBorder.border"), "Chat Room
Info", TitledBorder.LEADING, TitledBorder.TOP, null, new Color(0, 0,
0)));
            jPanel2.setBorder(BorderFactory.createTitledBorder("Online
Users"));

            this.jButton1.setText("Send");

            this.pack();
            this.setVisible(true);
```

```
            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension dim = tk.getScreenSize();
            int width = (int) (dim.width / 2.2);
            int height = (int) (dim.height / 2);
            this.setSize(width, height);

            int xPos = (dim.width / 2) - (this.getWidth() / 2);
            int yPos = (dim.height / 2) - (this.getHeight() / 2);
            this.setLocation(xPos, yPos);

            this.setTitle("Chat Room" + "--" +
this.clientConnect.getUsername());
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setVisible(true);

            ListenForWindow lForWindow = new ListenForWindow();
            this.addWindowListener(lForWindow);

            ListenForButton lForButton = new ListenForButton();
            this.jButton1.addActionListener(lForButton);

      }
      /**
       * Sends message to the server when the client disconnects the
server.
       * @author QiWang
       *
       */
      private class ListenForWindow extends WindowAdapter {
            public void windowClosing(WindowEvent e) {
                  try {
                        ClientChat.this.clientConnect.sendMsg("Client
leaves", 3);

                  } catch (Exception e1) {

                  }

            }
      }
      /**
       * Sends message to the server when the client sends chat
messages to other clients.
       * @author QiWang
       *
       */
      private  class ListenForButton implements ActionListener {

            @Override
            public void actionPerformed(ActionEvent e) {
                  String message = ClientChat.this.jTextArea3.getText();
                  ClientChat.this.jTextArea3.setText("");
                  ClientChat.this.clientConnect.sendMsg(message, 2);
            }

      }
```

```
	private class ListenForFocus extends FocusAdapter {
		public void focusGained(FocusEvent e)
	   {
			ClientChat.this.jTextArea3.setText("");
	   }
	}


}


package util;

import org.dom4j.io.*;
import org.dom4j.*;
import java.io.*;
import java.util.*;
/**
 * Contains several methods to read and build XML documents using
dom4j.
 * @author QiWang
 *
 */
public class XmlUtil {



	public static String getUserName(String xml) {


		try {
			SAXReader saxReader = new SAXReader();
			Document document = saxReader.read(new
StringReader(xml) );
			Element userName =
document.getRootElement().element("username");
			return userName.getText();
		} catch (DocumentException e) {
			e.printStackTrace();
		}
		return null;
	}

	public static String getLoginResponse(String xml) {
		String response = null;
		try {
			SAXReader saxReader = new SAXReader();
			Document document = saxReader.read(new
StringReader(xml));

			Element root = document.getRootElement();
			response = root.element("response").getText();

		} catch (DocumentException e) {
			e.printStackTrace();
		}
		return response;
```

```
	}

	public static String buildLoginMsg(String xml) {
		Document document = DocumentHelper.createDocument();
		Element rootElement = document.addElement("message");
		Element type =
document.getRootElement().addElement("type");
		type.setText("1");

		Element username = rootElement.addElement("username");
		username.setText(xml);
		return document.asXML();
	}

//	public static void buildUserList(Set<String> users) {
//		Document document = DocumentHelper.createDocument();
//
//	}

	public static String getType(String xml) {

		try {
			SAXReader saxReader = new SAXReader();
			Document document = saxReader.read(new
StringReader(xml) );
			Element type =
document.getRootElement().element("type");
			return type.getText();
		} catch (DocumentException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return null;
	}

	public static String getMsg(String xml) {

		try {
			SAXReader saxReader = new SAXReader();
			Document document = saxReader.read(new
StringReader(xml) );
			Element msgElement =
document.getRootElement().element("content");
			return msgElement.getText();
		} catch (DocumentException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		return null;
	}

	public static String buildServerMsg(String xml) {
		Document document = DocumentHelper.createDocument();
		Element rootElement = document.addElement("message");
		Element type = rootElement.addElement("type");
		type.setText("4");

		Element content = rootElement.addElement("content");
```

```
        content.setText(xml);
        return document.asXML();
    }

    public static String buildCloseServerWindowMsg() {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("5");
        return document.asXML();
    }

    public static String buildCloseClientWindowMsg(String username) {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("3");
        Element user = rootElement.addElement("username");
        user.setText(username);
        return document.asXML();
    }

    public static String buildLoginResponseMsg(String xml) {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("6");
        Element response = rootElement.addElement("response");
        response.setText(xml);
        return document.asXML();
    }

    public static String buildMsg(String username, String message) {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("2");
        Element user = rootElement.addElement("username");
        user.setText(username);
        Element response = rootElement.addElement("content");
        response.setText(message);
        return document.asXML();
    }

    public static String buildCloseClientWindowResponse() {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("7");
        return document.asXML();
    }

    public static String buildUserList(Set<String> users) {
        Document document = DocumentHelper.createDocument();
        Element rootElement = document.addElement("message");
        Element type = rootElement.addElement("type");
        type.setText("8");
        for (String name: users) {
```

```
                Element username = rootElement.addElement("username");
                username.setText(name);

            }
            return document.asXML();
      }

      public static List<String> getUserList(String xml) {
            List<String> names = new ArrayList<String>();

            try {
                SAXReader saxReader = new SAXReader();
                Document document = saxReader.read(new
StringReader(xml) );

                for (Iterator i =
document.getRootElement().elementIterator("username"); i.hasNext();) {
                    Element e = (Element) i.next();
                    names.add(e.getText());
                }
            } catch (DocumentException e) {
                e.printStackTrace();
            }
            return names;
      }

      public static String buildNonUserResponse() {
            Document document = DocumentHelper.createDocument();
            Element rootElement = document.addElement("message");
            Element type = rootElement.addElement("type");
            type.setText("9");
            return document.asXML();
      }

      public static String buildAddUser(String username) {
            Document document = DocumentHelper.createDocument();
            Element rootElement = document.addElement("message");
            Element name = rootElement.addElement("username");
            name.setText(username);
            Element type = rootElement.addElement("type");
            type.setText("10");
            return document.asXML();

      }

}


package util;

import java.io.Closeable;
import java.io.IOException;
/**
 * A class can be repeatedly used to close the input and out stream of
server and client socket.
 * @author QiWang
 *
 */
```

```
public class CloseUtil {
	public static void closeAll (Closeable... io) {
		for (Closeable temp: io) {

			try {
				if (temp != null) {
				temp.close();
				}
			} catch (IOException e) {
				e.printStackTrace();
			}

		}
	}

}
```