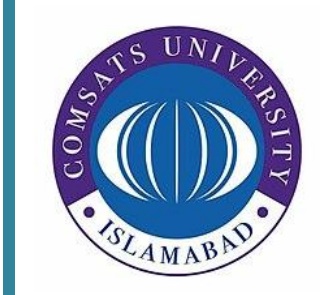


COMSATS UNIVERSITY ISLAMABAD
ATTOCK CAMPUS

Lab Report 6 : Operating System

Submitted to : Sir Fayyaz Ali

Group Members	Muaaz Shoaib FA20-BCS-074 Shahzeb Shaheen FA20-BCS-040

Rubrics Assessment Sheet for Operating System	
Lab #:	**Lab no 6**
Lab Title:	**Java based Client Server Chat Application**
Submitted by:	
Names **Muaaz Shoaib** **Shahzeb Shaheen**	**Registration** **FA20-BCS-074** **FA20-BCS-040**

Rubrics name & number		Marks	
		ln-Lab	Post lab
Engineering Knowledge	*R2:* Use of Engineering Knowledge and follow Experiment Procedures: *Ability to follow experimental procedure,control variables,and record Procedural steps on lab report.*		
Problem Analysis	*R6: Experimental Data Analysis :* *Ability to interept findings,compare them to values in the literature,identify weaknesses and limitations*		
Design	*RS: Best Coding Staudards:* *Ability lo follow the* coding standards and programming practices		
Modem Tools Usage	*R9: Undcrsta11d Tools: Ability to describe and explain the principles behind applicability of engineering tools*.		
Individual and Teamwork	*R9:Management of Team Work:* *Ability to appreciate, understand and work multidisciplinary team members*		

Rubrics #	R2	R6	RS	R9	Rl3
Jn -Lab					
Post- Lab					

Description :

Client Server Application :

This Application can be used for the purpose of Communication.

Post Lab Task :

To create a java based Client Server Chat Application by considering three Clients.

Coding :

Chat Client Class :

```
package edu.lmu.cs.networking;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JScrollPane;
```

```
import javax.swing.JTextArea;
import javax.swing.JTextField;

/**
 *  1. You have to write the code that prompt for Port and you enter the server port number.
 *  So you change this code
 *  /****Socket socket = new Socket(serverAddress, 9001);****
 *  private String getPort() {
      return JOptionPane.showInputDialog(
         frame,
         "Enter the Server Port:",
         "Port selection",
         JOptionPane.PLAIN_MESSAGE);
   }
 */
public class ChatClient {

   BufferedReader in;
   PrintWriter out;
   JFrame frame = new JFrame("Chatter");
   JTextField textField = new JTextField(40);
```

```
    JTextArea messageArea = new JTextArea(8, 40);
    /**
     * Constructs the client by laying out the GUI and registering a
     * listener with the textfield so that pressing Return in the
     * listener sends the textfield contents to the server.  Note
     * however that the textfield is initially NOT editable, and
     * only becomes editable AFTER the client receives the
NAMEACCEPTED
     * message from the server.
     */
    public ChatClient() {
        // Layout GUI
        textField.setEditable(false);
        messageArea.setEditable(false);
        frame.getContentPane().add(textField, "North");
        frame.getContentPane().add(new JScrollPane(messageArea),
"Center");
        frame.pack();

        // Add Listeners
        textField.addActionListener(new ActionListener() {
            /**
             * Responds to pressing the enter key in the textfield by sending
```

```
         * the contents of the text field to the server.    Then clear
         * the text area in preparation for the next message.
         */
        public void actionPerformed(ActionEvent e) {
            out.println(textField.getText());
            textField.setText("");
        }
    });
}

/**
 * Prompt for and return the address of the server.
 */
private String getServerAddress() {
    return JOptionPane.showInputDialog(
        frame,
        "Enter IP Address of the Server:",
        "Welcome to the Chatter",
        JOptionPane.QUESTION_MESSAGE);
}

/**
```

```
     * Prompt for and return the desired screen name.
     */
    private String getName() {
        return JOptionPane.showInputDialog(
            frame,
            "Choose a screen name:",
            "Screen name selection",
            JOptionPane.PLAIN_MESSAGE);
    }

    /**
     * Connects to the server then enters the processing loop.
     */
    private void run() throws IOException {

        // Make connection and initialize streams
        String serverAddress = getServerAddress();
        Socket socket = new Socket(serverAddress, 9001);
        in = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);
```

```
    // Process all messages from server, according to the protocol.
    while (true) {
        String line = in.readLine();
        if (line.startsWith("SUBMITNAME")) {
            out.println(getName());
        } else if (line.startsWith("NAMEACCEPTED")) {
            textField.setEditable(true);
        } else if (line.startsWith("MESSAGE")) {
            messageArea.append(line.substring(8) + "\n");
        }
    }
}

/**
 * Runs the client as an application with a closeable frame.
 */
public static void main(String[] args) throws Exception {
    ChatClient client = new ChatClient();
    client.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    client.frame.setVisible(true);
    client.run();
}
```

}

Chat Server Class :

```
package edu.lmu.cs.networking;

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.HashSet;

import java.io.File;

import java.io.FileWriter;


/**

 *     2. The server should do logging of your chat.

 */

public class ChatServer {


   /**

    * The port that the server listens on.

    */
```

```
    private static final int PORT = 9001;

    /**
     * The set of all names of clients in the chat room.  Maintained
     * so that we can check that new clients are not registering name
     * already in use.
     */
    private static HashSet<String> names = new HashSet<String>();

    /**
     * The set of all the print writers for all the clients.  This
     * set is kept so we can easily broadcast messages.
     */
    private static HashSet<PrintWriter> writers = new
HashSet<PrintWriter>();

    /**
     * The appplication main method, which just listens on a port and
     * spawns handler threads.
     */
    public static void main(String[] args) throws Exception {
        System.out.println("The chat server is running.");
```

```
    ServerSocket listener = new ServerSocket(PORT);
    try {
        while (true) {
            new Handler(listener.accept()).start();
        }
    } finally {
        listener.close();
    }
}



/**
 * A handler thread class.  Handlers are spawned from the listening
 * loop and are responsible for a dealing with a single client
 * and broadcasting its messages.
 */
private static class Handler extends Thread {
    private String name;
    private Socket socket;
    private BufferedReader in;
    private PrintWriter out;
```

```
        private BufferedReader bw;

/**
 * Constructs a handler thread, squirreling away the socket.
 * All the interesting work is done in the run method.
 */
public Handler(Socket socket) {
   this.socket = socket;
}

/**
 * Services this thread's client by repeatedly requesting a
 * screen name until a unique one has been submitted, then
 * acknowledges the name and registers the output stream for
 * the client in a global set, then repeatedly gets inputs and
 * broadcasts them.
 */
public void run() {
   try {

      // Create character streams for the socket.
      in = new BufferedReader(new InputStreamReader(
```

```
    socket.getInputStream()));
out = new PrintWriter(socket.getOutputStream(), true);

// Request a name from this client.  Keep requesting until
// a name is submitted that is not already used.  Note that
// checking for the existence of a name and adding the name
// must be done while locking the set of names.
while (true) {
    out.println("SUBMITNAME");
    name = in.readLine();
    if (name == null) {
        return;
    }
    synchronized (names) {
        if (!names.contains(name)) {
            names.add(name);
            break;
        }
    }
}

// Now that a successful name has been chosen, add the
```

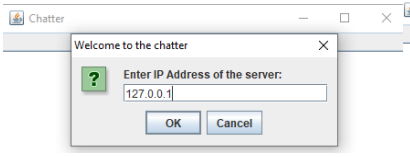
```
            // socket's print writer to the set of all writers so
            // this client can receive broadcast messages.
            out.println("NAMEACCEPTED");
            writers.add(out);

            // Accept messages from this client and broadcast them.
            // Ignore other clients that cannot be broadcasted to.
            while (true) {
                String input = in.readLine();
                if (input == null) {
                    return;
                }
                for (PrintWriter writer : writers) {
                    writer.println("MESSAGE " + name + ": " + input);
                }
            }
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            // This client is going down!  Remove its name and its print
            // writer from the sets, and close its socket.
            if (name != null) {
```

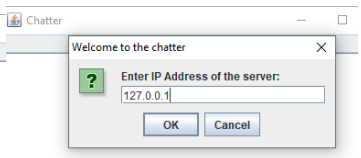
```
                names.remove(name);
            }
            if (out != null) {
                writers.remove(out);
            }
            try {
                socket.close();




            } catch (IOException e) {
            }
        }
    }
  }
}
```

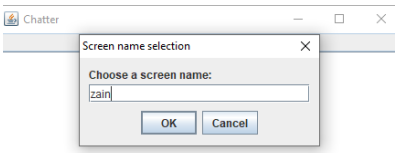
Screenshots :

```
            socket.getInputStream()));
        out = new PrintWriter(socket.getOutputStream(), true);

        // Request a name from this client.  Keep requesting until
        // a name is submitted that is not already used.  Note that
        // checking for the existence of a name and adding the name
        // must be done while locking the set of names.
        while (true) {
            out.println("SUBMITNAME");
            name = in.readLine();
            if (name == null) {
                return;
            }
            synchronized (names) {
                if (!names.contains(name)) {
                    names.add(name);
                    break;
                }
            }
        }

        // Now that a successful name has been chosen, add the
        // socket's print writer to the set of all writers so
        // this client can receive broadcast messages.
        out.println("NAMEACCEPTED");
        writers.add(out);

        // Accept messages from this client and broadcast them.
        // Ignore other clients that cannot be broadcasted to.
        while (true) {
            String input = in.readLine();
            if (input == null) {
                return;
```

```
69                  out.println(textField.getText());
70                  textField.setText("");
71              }
72          });
73      }
74
75      /**
76       * Prompt for and return the address of the server.
77       */
78      private String getServerAddress() {
79          return JOptionPane.showInputDialog(
80              frame,
81              "Enter IP Address of the Server:",
82              "Welcome to the Chatter",
83              JOptionPane.QUESTION_MESSAGE);
84      }
85
86      /**
87       * Prompt for and return the desired screen name.
88       */
89      private String getName() {
90          return JOptionPane.showInputDialog(
91              frame,
92              "Choose a screen name:",
93              "Screen name selection",
94              JOptionPane.PLAIN_MESSAGE);
95      }
96
97      /**
98       * Connects to the server then enters the processing loop.
99       */
100     private void run() throws IOException {
```







zain: hi
zain: how are you waleed and ubaid