**Implementation of concurrent chat server that allows current logged in users to  
communicate one with other.**

**Program:**

// **Java implementation of Server side**

// It contains two classes : Server and ClientHandler

// Save file as Server.java

import java.io.\*;

import java.util.\*;

import java.net.\*;

// Server class

public class Server

{

// Vector to store active clients

static Vector<ClientHandler> ar = new Vector<>();

// counter for clients

static int i = 0;

public static void main(String[] args) throws IOException

{

// server is listening on port 1234

ServerSocket ss = new ServerSocket(1234);

Socket s;

// running infinite loop for getting

// client request

while (true)

{

// Accept the incoming request

s = ss.accept();

System.out.println("New client request received : " + s);

// obtain input and output streams

DataInputStream dis = new DataInputStream(s.getInputStream());

DataOutputStream dos = new DataOutputStream(s.getOutputStream());

System.out.println("Creating a new handler for this client...");

// Create a new handler object for handling this request.

ClientHandler mtch = new ClientHandler(s,"client " + i, dis, dos);

// Create a new Thread with this object.

Thread t = new Thread(mtch);

System.out.println("Adding this client to active client list");

// add this client to active clients list

ar.add(mtch);

// start the thread.

t.start();

// increment i for new client.

// i is used for naming only, and can be replaced

// by any naming scheme

i++;

}

}

}

// ClientHandler class

class ClientHandler implements Runnable

{

Scanner scn = new Scanner(System.in);

private String name;

final DataInputStream dis;

final DataOutputStream dos;

Socket s;

boolean isloggedin;

// constructor

public ClientHandler(Socket s, String name,

DataInputStream dis, DataOutputStream dos) {

this.dis = dis;

this.dos = dos;

this.name = name;

this.s = s;

this.isloggedin=true;

}

@Override

public void run() {

String received;

while (true)

{

try

{

// receive the string

received = dis.readUTF();

System.out.println(received);

if(received.equals("logout")){

this.isloggedin=false;

this.s.close();

break;

}

// break the string into message and recipient part

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

String MsgToSend = st.nextToken();

String recipient = st.nextToken();

// search for the recipient in the connected devices list.

// ar is the vector storing client of active users

for (ClientHandler mc : Server.ar)

{

// if the recipient is found, write on its

// output stream

if (mc.name.equals(recipient) && mc.isloggedin==true)

{

mc.dos.writeUTF(this.name+" : "+MsgToSend);

break;

}

}

} catch (IOException e) {

e.printStackTrace();

}

}

try

{

// closing resources

this.dis.close();

this.dos.close();

}catch(IOException e){

e.printStackTrace();

}

}

}

**// Java implementation for multithreaded chat client**

**// Save file as Client.java**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class Client

{

final static int ServerPort = 1234;

public static void main(String args[]) throws UnknownHostException, IOException

{

Scanner scn = new Scanner(System.in);

// getting localhost ip

InetAddress ip = InetAddress.getByName("localhost");

// establish the connection

Socket s = new Socket(ip, ServerPort);

// obtaining input and out streams

DataInputStream dis = new DataInputStream(s.getInputStream());

DataOutputStream dos = new DataOutputStream(s.getOutputStream());

// sendMessage thread

Thread sendMessage = new Thread(new Runnable()

{

@Override

public void run() {

while (true) {

// read the message to deliver.

String msg = scn.nextLine();

try {

// write on the output stream

dos.writeUTF(msg);

} catch (IOException e) {

e.printStackTrace();

}

}

}

});

// readMessage thread

Thread readMessage = new Thread(new Runnable()

{

@Override

public void run() {

while (true) {

try {

// read the message sent to this client

String msg = dis.readUTF();

System.out.println(msg);

} catch (IOException e) {

e.printStackTrace();

}

}

}

});

sendMessage.start();

readMessage.start();

}

}

