CSCE 416

Run ConfServer and then run confclient

Chart applicable – where chart one specific type

**ConfServer**

* Keep track of all the clients
* One class: does all conference server jobs
* Have parent class: should allow any clients to connect to me should be able to read from all clients & welcome socket to accept new client, watch client sockets for images they send
* Right now have 11 sockets bc tcp then have 10 clients:
  + 1 way to do: to watch can make a select call watch all sockets and if input on any of them let me know
  + Another way: keep calling them, called busy loop don’t want to do that
  + Using threads is another approach: what we are doing: 11 sockets so need 11 threads
* When thread blocked other should still be able to run
* Main thread: parent thread, code in main, parent thread watching the welcome socket to accept new connections, watching/blocking/waiting welcome socket to make a new thread and do so again
* Any time get new connection start new thread
* Child thread: running method
  + Watch socket any time gets something send message
  + Reading from client socket determined by accept call
* “implements Runnable”: thread will be created using an object of this class through implements runnable
  + New thread created for new object of this class (instance of this class)
* Keeping track of ‘clientSock’
  + Each thread has an. Instance of clientSock
* Dynamically Keep track of active client
  + So can send all clients any time one sends a message
  + Static List<PrintWriter> clientList
  + Only just one list^ for all clients – not a member of an instance, that’s why static is important, without everyone would have their own list
* addClient method:
  + when multiple working concurrently, when multiple threads working on something its important to lock it & unlock it so see edits (idea of atomic)
  + to ensure sharing is done safely by locking, order does not matter one just has to go first
  + to make sure adding is atomic but keyword: ‘synchronized’ and java takes care of blocking others and letting one go at a time
* removeClient method
  + same way as add
  + dynamic, come & go
  + needs to be static and synchronized
  + pause printWriter to remove
  + Boolean for if able to remove or not true or false
  + No partial jobs, we do not care about order
* relayMessage method
  + job if get message is to relay it to everyone
  + iterating through PrintWriter list and printing it
* run method
  + whenever say implements Runnable expected that has run method, this method
  + when parent creates a thread then threads starts this is what it does
  + child is responsible for reading messages and sending it to everybody
  + comes here after start
  + preparing to read from socket
  + need client to give their name
  + so first thing client does is connect to server and write the name of the user
  + clientName: reader name
  + toClientWriter: PrintWriter is very similar to system.out
    - system.out prints to the screen
    - here it prints to the socket
    - auto flush on: anytime write to new line automatically, send to socket immediately, not buffering it, that’s why it set to ‘true’
  + add client to list of client, list has been updated – because used synchronized don’t need to worry about multiple threads updating client list simultaneously
  + while loop: keep doing till client end of file (EOF), keep doing until client closes connection
    - in terminal when client types: control D, done writing that’s when we get end of file
    - end of file also when gets a null when reading line means client closed connection
    - if open relaying the message by writing the line
  + when come out of while loop done with client and remove client
  + and then need to close everything that means done
* main method:
  + parent thread: responsible for accepting new connections
  + where program starts when run this class, where get arguments
  + expect the port number: the argument for the server, port given as an argument
  + in try clause:
    - create server socket
    - initialize clientList (initially empty)
    - while: infinite loop: main thread just waiting here, when get new client create ConfServer object – and then take that object and create a thread out of that object
    - Then say start: essentially calling run when say child.start()
    - Then after start parent thread
  + Put catch because the server socket can throw exceptions so we catch them
  + nothing special about calling it child
  + we want server living forever, unless killed but really don’t want that

**ConfClient**

* read from the keyboard and read from the socket
* have two threads because watching two things^
* parent thread: read from socket and displaying it
* child thread: read from keyboard and send from socket
* since still a thread need the Runnable
* forUserReader: read from keyboard, BufferedReader to read a line
* toSockWriter: printing, PrintWriter to print a line
* parent thread doesn’t need a object, child thread does & needs ^ 2 objects
* run method:
  + try method: to read and write
  + while loop: infinite loop until user types EOF (type Control D)
  + if null done if read nothing from user
  + otherwise write message to socket and go back to reading
* main method:
  + read from the keyboard and socket use BufferedReader
  + to write to display use system.out.
  + to write to socket use PrintWriter
  + client needs server info
    - expecting 3 arguments: server host(ip or name), server’s port, name of client and then passing that to server
  + creating as socket to connect to server
  + Try: setting up thread to read from user and send to server
    - create toSockWriter from the output stream
    - server is reading my name so it can put for all the messages I send
    - create reader from keyboard
    - create conference client object (thread) using two arguments: reader and writer, using object create thread so when say start goes to run method
  + try: read from server and display user
    - fromSockReader: so can read from sockey
    - create infinite loop to read from socket until null

**Running in Terminal**

* Java ConfServer 8000
* Diff window: java ConfClient localhost 8000 alice
* Diff window: java ConfClient localhost 8000 bob
* Diff window: java ConfClient localhost 8000 david
* Should get notifications on Server that clients joined the conference
* On a client: type: hello can you hear me
  + Everyone will receive client name: their text
* \

Research:

I write these types of server applications frequently, as a way of connection client through a relay server when direct connections between them are not possible. The solution is simple, put each socket that you receive from the accept() function into a List or Tree of your choice.

List<Socket> connectionList = new ArrayList<Socket>();

...

Socket sock =sersock.accept();

connectionList.add(sock);

...

for (Socket connection : connectionList) {

connection.getOutputStream().write(msgBytes);

}

<https://www.nakov.com/books/inetjava/source-code-html/Chapter-1-Sockets/1.4-TCP-Sockets/TCPForwardServer.java.html>

<http://pirate.shu.edu/~wachsmut/Teaching/CSAS2214/Virtual/Lectures/chat-client-server.html>

how to keep track of active clients:

static Collection<Socket> activeSockets = new ConcurrentLinkedQueue<>();

/\*for(int i =0; i < numClients; i++ )

{

//get or iterator

System.out.println(clientList.get(i));

}\*/

**public** **class** chat\_server **implements** Runnable

{

//each thread has an instance of a socket

**private** Socket cSock;

//dynamic list of clients: name, state & sockey

**private** **static** HashMap<String,PrintWriter> clientList;

//hashmap for connections containing name of both clients

**private** **static** HashMap<String,String> connectTable;

//counter for clients

**private** **static** **int** numClients = 100;

//constructor

**public** chat\_server(Socket sock)

{

cSock = sock;

numClients =0;

}

//atomic add a client to client list

**public** **static** **synchronized** **boolean** addClient(String name, PrintWriter toClientWriter)

{

clientList.put(name, toClientWriter);

numClients++;

**return**(clientList.containsValue(toClientWriter));

}

//remove the client

**public** **static** **synchronized** **boolean** removeClient(String name, PrintWriter toClientWriter)

{

numClients--;

**return**(clientList.remove(name, toClientWriter));

}

**public** **static** String compoundKey(String name, String state)

{

**return** name + " " + state;

}

//what this to rely message to a specific client not the whole client list

**public** **static** **synchronized** **void** relayMessage(String mesg)

{

// Iterate through the list and send message to each client

//for (PrintWriter clientWriter : clientList)

**for** (String msg : clientList.keySet())

System.out.println(mesg);

//clientWriter.println(mesg);

}

//add to connection table

/\*public static synchronized void connectClients()

{

}\*/

**public** **static** **void** main(String args[])

{

// Server needs a port to listen on

**if** (args.length != 1) {

System.out.println("usage: java chat server <port>");

System.exit(1);

}

// Be prepared to catch socket related exceptions

**try** {

// Create a server socket with the given port

ServerSocket serverSock =

**new** ServerSocket(Integer.parseInt(args[0]));

System.out.println("Waiting for clients ...");

// Keep track of active clients

clientList = **new** HashMap<String,PrintWriter>();

// Keep accepting/serving new clients

**while** (**true**) {

// Wait for another client

Socket clientSock = serverSock.accept();

// Spawn a thread to read/relay messages from this client

Thread child = **new** Thread(**new** chat\_server(clientSock));

child.start();

}

}

**catch**(Exception e) {

System.out.println(e);

}

}

// The child thread starts here

**public** **void** run()

{

// Read from the client and relay to other clients

**try** {

// Prepare to read from socket

BufferedReader fromClientReader = **new** BufferedReader(

**new** InputStreamReader(cSock.getInputStream()));

// Get the client name

String clientName = fromClientReader.readLine();

System.out.println("Client Accepted");

// SOCKET: Prepare to write to socket (client screen) with auto flush on

PrintWriter toClientWriter =

**new** PrintWriter(cSock.getOutputStream(), **true**);

String state = "free";

// Add this client to the active client list

addClient(compoundKey(clientName, state),toClientWriter);

//print client list with state

System.out.println("List of clients and states");

toClientWriter.println("List of clients and states");

//for loop to list the client name and state

**for**( Map.Entry<String, PrintWriter> entry : clientList.entrySet() ){

System.out.println( entry.getKey()); //+ " " + entry.getValue() );

toClientWriter.println(entry.getKey());

}

//connecting to clients

toClientWriter.println("Connect to which client?");

String userChoice = fromClientReader.readLine();

**if**(clientList.getKey().startsWith(userChoice) == userChoice)//clientList.getKey().startsWith("xxxx")

{

toClientWriter.println("found person");//connectClients(clientName, userChoice);

}

**else**

{

toClientWriter.println("failed");

}

// Keep doing till client sends EOF

**while** (**true**) {

// Read a line from the client

String line = fromClientReader.readLine();

// If we get null, it means client sent EOF

**if** (line == **null**)

**break**;

// Send the line to all active clients

relayMessage(clientName + ": " + line);

}

// Remove this client from active list

removeClient(clientName, toClientWriter);

// Done with the client, close everything

toClientWriter.close();

System.out.println(clientName + " left the conference");

}

**catch** (Exception e) {

System.out.println(e);

}

}

}

//Jhada Kahan-Thomas

//I/O stuff

import java.io.\*;

//socket classes

import java.net.\*;

//list usage

import java.util.\*;

public class chat\_server implements Runnable

{

//each thread has an instance of a socket

private Socket cSock;

//dynamic list of clients: name, state & sockey

private static HashMap<PrintWriter,String[]> clientList;

private static String[] list = new String[2]; //{"name", "status"};

//hashmap for connections containing name of both clients

private static HashMap<String,String> connectTable;

//counter for clients

private static int numClients = 100;

//constructor

public chat\_server(Socket sock)

{

cSock = sock;

numClients =0;

}

//atomic add a client to client list

public static synchronized boolean addClient(PrintWriter toClientWriter, String [] list)

{

clientList.put(toClientWriter, list);

numClients++;

return(clientList.containsKey(toClientWriter));

}

//remove the client

public static synchronized boolean removeClient(PrintWriter toClientWriter, String[] list)

{

numClients--;

return(clientList.remove(toClientWriter, list));

}

//establish a connection

//connecting to clients

public static PrintWriter connect(BufferedReader fromClientReader,PrintWriter toClientWriter)

{

toClientWriter.println("Connect to which client?");

String userChoice = fromClientReader.readLine();

for(Map.Entry<PrintWriter, String[]> temp : clientList.entrySet()){

if(temp.getValue()[0].equalsIgnoreCase(userChoice))

{

PrintWriter peerSoc = new PrintWriter(temp.getKey());

acceptCon(toClientWriter, peerSoc);

toClientWriter.println("You are connected to " + userChoice);//connectClients(clientName, userChoice);

return peerSoc;

}

}

return null;

}

//connection attempting to be estabilished

public static synchronized boolean acceptCon(PrintWriter requester, PrintWriter cSoc)

{

try {

// Prepare to read from socket

BufferedReader fromClientReader = new BufferedReader(new InputStreamReader(cSoc.getInputStream()));

// SOCKET: Prepare to write to socket (client screen) with auto flush on

PrintWriter toClientWriter = new PrintWriter(cSoc);

toClientWriter.println("Received request from " + clientList.get(requester)[0]

+ "\n Connect? y or n");

String response = fromClientReader.readLine();

if(response.equals('yes'))

{

return true;

}

else

return false;

}

catch(Exception e) {

System.out.println(e);

}

}

//what this to rely message to a specific client not the whole client list

public static synchronized void relayMessage(String mesg, PrintWriter peer)

{

// Iterate through the list and send message to each client

//for (PrintWriter clientWriter : clientList)

for(PrintWriter clientWriter : clientList.keySet())

{

peer.println(mesg);

//return;

}

//System.out.println(mesg);

//clientWriter.println(mesg);

}

//add to connection table

/\*public static synchronized void connectClients()

{

}\*/

public static void main(String args[])

{

// Server needs a port to listen on

if (args.length != 1) {

System.out.println("usage: java chat server <port>");

System.exit(1);

}

// Be prepared to catch socket related exceptions

try {

// Create a server socket with the given port

ServerSocket serverSock =

new ServerSocket(Integer.parseInt(args[0]));

System.out.println("Waiting for clients ...");

// Keep track of active clients

clientList = new HashMap<PrintWriter,String[]>();

// Keep accepting/serving new clients

while (true) {

// Wait for another client

Socket clientSock = serverSock.accept();

// Spawn a thread to read/relay messages from this client

Thread child = new Thread(new chat\_server(clientSock));

child.start();

}

}

catch(Exception e) {

System.out.println(e);

}

}

// The child thread starts here

public void run()

{

// Read from the client and relay to other clients

try {

// Prepare to read from socket

BufferedReader fromClientReader = new BufferedReader(

new InputStreamReader(cSock.getInputStream()));

// Get the client name

String clientName = fromClientReader.readLine();

System.out.println("Client Accepted");

// SOCKET: Prepare to write to socket (client screen) with auto flush on

PrintWriter toClientWriter =

new PrintWriter(cSock.getOutputStream(), true);

String state = "free";

String[] info = new String[]{clientName,state};

// Add this client to the active client list

addClient(toClientWriter,info);

//print client list with state

System.out.println("List of clients and states");

toClientWriter.println("List of clients and states");

//for loop to list the client name and state

for( Map.Entry<PrintWriter, String[]> temp : clientList.entrySet() ){

System.out.println(temp.getValue()[0] + " "+ temp.getValue()[1]);//(toClientWriter)[0] + temp.get(toClientWriter)[1]);

toClientWriter.println(temp.getValue()[0]+ " " + temp.getValue()[1]);

}

/\*//connecting to clients

toClientWriter.println("Connect to which client?");

String userChoice = fromClientReader.readLine();

for(Map.Entry<PrintWriter, String[]> temp : clientList.entrySet()){

if(temp.getValue()[0].equalsIgnoreCase(userChoice))

{

PrintWriter peerSoc = new PrintWriter(temp.getKey());

acceptCon(toClientWriter, peerSoc);

toClientWriter.println("You are connected to " + userChoice);//connectClients(clientName, userChoice);

\*/

// Keep doing till client sends EOF

PrintWriter peerSoc = new PrintWriter(connect());

while (true) {

// Read a line from the client

String line = fromClientReader.readLine();

// If we get null, it means client sent EOF

if (line == null)

break;

// Send the line to all active clients

relayMessage(clientName + ": " + line + "\n", peerSoc);

}

//}

/\*else

{

toClientWriter.println("no client found");

}\*/

// Remove this client from active list

removeClient(toClientWriter,info);

// Done with the client, close everything

toClientWriter.close();

System.out.println(clientName + " left the conference");

}

catch (Exception e) {

System.out.println(e);

}

import java.io.\*;

//socket classes

import java.net.\*;

//list usage

import java.util.\*;

public class chat\_server implements Runnable

{

//each thread has an instance of a socket

private Socket cSock;

//dynamic list of clients: name, state & sockey

private static HashMap<PrintWriter,String[]> clientList;

private static ArrayList<String> collection = new ArrayList<String>(); //{"name", "status"};

//hashmap for connections containing name of both clients

private static HashMap<String,String> connectTable;

//counter for clients

private static int maxClients = 100;

//constructor

public chat\_server(Socket sock)

{

cSock = sock;

}

//atomic add a client to client list

public static synchronized boolean addClient(PrintWriter toClientWriter, String [] list)

{

clientList.put(toClientWriter, list);

return(clientList.containsKey(toClientWriter));

}

//remove the client

public static synchronized boolean removeClient(PrintWriter toClientWriter, String[] list)

{

return(clientList.remove(toClientWriter, list));

}

//what this to rely message to a specific client not the whole client list

public static synchronized void relayMessage(String mesg, PrintWriter peer)

{

// Iterate through the list and send message to each client

//for (PrintWriter clientWriter : clientList)

for(PrintWriter clientWriter : clientList.keySet())

{//might need an if

peer.println(mesg);

//return;

}

//System.out.println(mesg);

//clientWriter.println(mesg);

}

// Since all threads share this, we use "synchronized" to make it atomic

public static synchronized void relayStatus()

{

// Iterate through the list and send message to each client

for (PrintWriter clientWriter : clientList.keySet())

{

for(Map.Entry<PrintWriter, String[]> temp : clientList.entrySet())

{

clientWriter.println(temp.getValue()[0]+ " " + temp.getValue()[1]);

}

}

}

//add to connection table

/\*public static synchronized void connectClients()

{

}\*/

public static void main(String args[])

{

// Server needs a port to listen on

if (args.length != 1) {

System.out.println("usage: java chat server <port>");

System.exit(1);

}

// Be prepared to catch socket related exceptions

try {

// Create a server socket with the given port

ServerSocket serverSock =

new ServerSocket(Integer.parseInt(args[0]));

System.out.println("Waiting for clients ...");

// Keep track of active clients

clientList = new HashMap<PrintWriter,String[]>();

// Keep accepting/serving new clients

while (true) {

// Wait for another client

Socket clientSock = serverSock.accept();

for(int i = 0; i < maxClients; i++)

{

// Spawn a thread to read/relay messages from this client

Thread child = new Thread(new chat\_server(clientSock));

child.start();

break;

}

}

}

catch(Exception e) {

System.out.println(e);

}

}

// The child thread starts here

public void run()

{

// Read from the client and relay to other clients

try {

// Prepare to read from socket

BufferedReader fromClientReader = new BufferedReader(

new InputStreamReader(cSock.getInputStream()));

// Get the client name

String clientName = fromClientReader.readLine();

System.out.println("Client Accepted");

// SOCKET: Prepare to write to socket (client screen) with auto flush on

PrintWriter toClientWriter =

new PrintWriter(cSock.getOutputStream(), true);

String state = "free";

String[] info = new String[]{clientName,state};

// Add this client to the active client list

addClient(toClientWriter,info);

System.out.println("List of clients and states");

toClientWriter.println("List of clients and states");

for( Map.Entry<PrintWriter, String[]> temp : clientList.entrySet() )

System.out.println(temp.getValue()[0] + " "+ temp.getValue()[1]);

relayStatus();

String userChoice = fromClientReader.readLine();

//WANT a user

PrintWriter requestedSock = new PrintWriter(toClientWriter);

String sender = "";

String conn = "n";

//toClientWriter.println("Connect to which client?");

//String userChoice = fromClientReader.readLine();

for(Map.Entry<PrintWriter, String[]> temp : clientList.entrySet()){

if(temp.getValue()[0].equalsIgnoreCase(userChoice))

{

requestedSock= temp.getKey();

//their name goes

}

}

//check for waiting requests

if(requestedSock == toClientWriter)

{

toClientWriter.println("You got a request from " + clientName

+ "\n Connect? y or n");

conn = fromClientReader.readLine();

}

if(conn.equals("y"))

{

while (true) {

state = "busy";

// Read a line from the client

String line = fromClientReader.readLine();

// If we get null, it means client sent EOF

if (line == null)

break;

// Send the line to all active clients

relayMessage(clientName + ": " + line + "\n", requestedSock);

}

}

/\*request(toClientWriter,)

//received a request

toClientWriter.println("Received")

do you want to

y or no

return of request

relay to client

//request fullfilled

toClientWriter.println("You are connected to "

relay to client

//connecting to clients

//toClientWriter.println("Connect to which client?");

//String userChoice = fromClientReader.readLine();

for(Map.Entry<PrintWriter, String[]> temp : clientList.entrySet()){

if(temp.getValue()[0].equalsIgnoreCase(userChoice))

{

PrintWriter peerSoc = new PrintWriter(temp.getKey());

acceptCon(toClientWriter, fromClientReader, peerSoc);

toClientWriter.println("You are connected to " + userChoice);//connectClients(clientName, userChoice);

// Keep doing till client sends EOF

PrintWriter peerSoc = new PrintWriter(connect());

while (true) {

// Read a line from the client

String line = fromClientReader.readLine();

// If we get null, it means client sent EOF

if (line == null)

break;

// Send the line to all active clients

relayMessage(clientName + ": " + line + "\n", peerSoc);

}

//}

else

{

toClientWriter.println("no client found");

}\*/

// Remove this client from active list

removeClient(toClientWriter,info);

// Done with the client, close everything

toClientWriter.close();

System.out.println(clientName + " left the conference");

}

catch (Exception e) {

System.out.println(e);

}

}

}