CS440 Lab 2

Alec Waichunas
Angel Galeana
April 4th, 2019
Prof. Bidyut Gupta

Introduction

This client server application allows multiple clients to connect to a server. Once a client is connected to the server, they can create chat rooms with other connected clients. The server controls the flow of the clients, and the chat rooms. The client will just connect to the server and send/receive messages to communicate with the server or other clients.

Execution

The Server must be compiled to run. To compile the server, you can use any Java supported IDE to immediately start it, otherwise you can compile it through a CLI with javac and run it with java.

The Client must be compiled to run. To compile the client, you can use any Java supported IDE to immediately start it, otherwise you can compile it through a CLI with javac and run it with java. The server must be compiled and runned before the client can be runned. Otherwise errors will occur.

The Server Program

The Server Program consists of 4 classes. Each with their own purpose to connect to the client, and accept different commands through it.

The entry class of the program is the server class. It creates a server socket using Java's net API. The server then calls an init method, which goes into a loop that will accept client connections, assign them an ID and create a Client object.

```
while(!server.isClosed()){
 try{
  Socket clientSocket = server.accept();
  int id = this.getNewClientID();
  Client client = new Client(id, clientSocket, this);
  clients.add(id, client);
 }catch(IOException e){
  //could not connect to client
  e.printStackTrace();
 }
}
```

After the client is accepted and created the client class starts up. In the client's class constructor it creates an object called ClientInput, which is another class. This class creates a seperate thread and listens to the client socket for incoming connections.

```
String data;
try{
 while((data = input.readLine()) != null)
  client.readClient(data);
 }catch(IOException e){
  e.printStackTrace();
}
```

Once the client has sent a message and the readClient method is ran in the Client class, the program checks to see if the client has a chatroom. If the client is in a chatroom the message will be sent to the the other clients through the ChatRoom object, otherwise the message is tested for being a command.

```
if(chatRoom == null){
 //looking for command
 String[] commands = message.split(" ");
 if(commands[0].equalsIgnoreCase("talk")){
  //creates a chat room with one other client
  .
  .
  .
  this.chatRoom = new ChatRoom(clients);
 }else if(commands[0].equalsIgnoreCase("conference") &&
  commands[1].equalsIgnoreCase("talk")){
  //create a chat room with multiple clients
  .
  .
  .
  this.chatRoom = new ChatRoom(clients);
 }else if(commands[0].equalsIgnoreCase("accept")){
  //check to see if chatroom/client exists and accept request
  String[] clientinfo = commands[1].split("@");
  Client reqclient = this.server.getClientByIpAndName(clientinfo[0],
clientinfo[1]);
  if(reqclient == null || reqclient.chatRoom == null){
   this.sendToClient("Could not find " + clientinfo[0] + "'s Chat Room!'");
  }else{
   reqclient.chatRoom.acceptRequest(this);
  }
 }
}else{
 //the only command during a chat
 if(message.equals("Exit"))
  chatRoom.clientDisconnect(this);
 else
  //send message if it is not a command
  chatRoom.castMessage(this, message);
}
```

If the client creates a chatroom with one or multiple clients, it will create a new chatroom object from the ChatRoom class, that will send a message to the other clients. The other clients will not be assigned the chatroom until they accept the connection. Once the client has accepted the connection and sends a message the chatroom class will cast the message to the other clients.

```
for(Client c : this.clients){
 if(!c.equals(client) && c.getChatRoom().equals(this))
  c.sendToClient(client.getName() + ": " + message);
}
```

The Client Program

The client program consist of one class using the Runnable interface. This class consisist of 4 methods.

The entry method creates the socket and starts the thread. PrintWriter is used to translate objects into text-output. Then makes a call to the ReadClient() method.

```
private Socket s;
public client() throws IOException {
 s = new Socket("localhost", 12252);
 PrintWriter pr = new PrintWriter(s.getOutputStream());
 (new Thread(this)).start();
 ReadClient(pr);
}
```

This method reads input from the console using a scanner and sends ot to the server. The while loops checks that the socket is open.

```
public void ReadClient(PrintWriter pr) {
 Scanner scan = new Scanner(System.in);
 while(!s.isClosed()){
        String line = scan.nextLine();
        pr.println(line);
        pr.flush();
 }
}
```

Next, the run() method reads lines from the server and displays them to the console. There is a while loop present here to check that the socket is still open.

```
public void run() {
 while(!s.isClosed()){
  try {
   InputStreamReader in = new InputStreamReader(s.getInputStream());
         BufferedReader bf = new BufferedReader(in);
```

```
            String str = bf.readLine();
            System.out.println(str);
           }catch(IOException e) {
                   System.err.println("Could not send message");
           }

    }
   }
```

Finally, the main method initializes a new client to run the program.

```
public static void main(String[] args) throws IOException {
 client cl1 = new client();
}
```

The output of the client is what the server sends over, and what the client inputs. An example is as follows below:

Client0 Terminal: You are Client0 Talk request fromClient1@127.0.0.1. Respond with "accept Client1@127.0.0.1" accept Client1@127.0.0.1 test Client1: cool cool beans Client1: omege cool beans Client1: Connection Terminated with Client1

Client1 Terminal: You are Client1 talk client0 Ringing Client0 Client0: Talk connection established with Client0 Client0: test cool Client0: cool beans omege cool beans Exit

Error detection within the programs

The server will check if the client exists when a command is ran, if it does not exist the output will look like a version of below.

```
You are Client0
talk client1
Client1 not logged in!
conference talk client1 client2
Client1 not logged in!
conference talk client2 client3
Client2 not logged in!
talk client2
Client2 not logged in!
```

The chat room will not be made if one of the clients are not logged in.

Known Bugs

While testing the program, no bugs were found and the program needs to be further stress tested to figure out the unknown bugs.

Packages

There are a few java packages that were used to create this program.
The net package was used to let the server listen on a port, and for the client to connect to the server.
The io package was used to read and write data from the client and server.
The util package was used to create linked list to create track of the clients in the server.

Possible Improvements

One improvement to the program, would to have the client be in control of the chat room more. The server takes control of the entire program, and the client is simply just an intereface for it.

Conclusion

The lab opened up the oppurtunity to easily install a system where clients can create chat rooms with each other easily. Having the client object in the server have reference to the chat room object made sending messages to the other clients very easy.