Stefano Mich

Assignment 3: Java TCP Chat

1) The program is divided in 2 classes, Client and Server.
   The Server class is responsible for starting the server and mediating between different clients. First of all the server needs to be started and the user can choose the port number where to open it. Then the server opens a socket at a certain port and it's ready to receive connections to that specific port.
   Once the server is opened, a client can connect to it by opening a socket with the specific address of the server(“localhost” in our case) and with the port number. The client also opens an ObjectInputStream for incoming objects and ObjectOutputStream for outgoing ones.
   To send messages to the server, the client uses the ObjectOutputStream and for receiving messages from the server it uses the ObjectInputStream. The server works with the same procedure.
   Every time a new client connects to the server, a ClientThread is opened for him in the server and it remains opened until the client logs out. The thread is responsible for buffering incoming commands and messages from the client and for sending messages back to it.
   The client also opens his own thread, used to listen for incoming messages.

   HOW TO
   To run the program, simply run the Server Class in a terminal page and then run the Client class as many times as you prefer in different terminal pages. When started, the Server will wait for Clients to connect. When the client are connected, then they can directly type a message that will be displayed on the server and on the other client terminals.
   To close the server use the CTRL+C command and from the client to log out use the LOGOUT command.

   BUGS AND PROBLEMS

- Unfortunately I didn't find a way to buffer incoming messages and print them only when the client has finished typing. This is of course a really important feature for a live chat. In my case, if a client is typing and in the same time he receives a message, then the message he was typing get lost. With my design, I was not able to find a way to solve this problem. I should have thought about it in the early stage, before coding.
  I thought about a method in which if the command line of the client is void, then he could receive messages, otherwise the message would be stored in a string array and then printed, but I was not able to make it work.
- If a client logs out by typing CTRL+C, the server displays an exception, because it's trying to interpret the message of the user, that instead has just logged out.
- If the server is disconnected and a client tries to send a message, a SocketException is displayed saying that the pipe is broken.

UDP VERSION

Since the udp protocol is connection-less, no thread would be possible between the client and the server. The server, instead of listening for incoming clients, will listen for incoming messages.

The client will send messages as packets and the server will buffer them and then send them back to the clients as packets.

Usually the udp version is faster than the tcp one because it does not require the handshaking and other procedures, but it requires a checksum to control if a packet has been lost. Said this, a chat written in Java is really fast if not performed with many clients and tough the differences between the TCP and the UDP versions would not be visible.