PA3: Extending Functionality of Multi-threaded Chat Server & Client

Sanidhya Singal

The assignment submission consists of 2 java files: `Server_2015085r1.java` and `Client_2015085r1.java`. As is clear by the names, the former file implements a multi-threaded server while the latter implements a client. Multiple clients can be created by creating multiple instances of clients using the latter file.
The default port for server and client is `1222`. In case you don't specify the port no., the server runs on port `1222`. Also, the default host is `localhost`. The client runs on `localhost` and port `1222` by default.
The class used to create multiple threads for handling multiple clients is private and is written in `Server_2015085r1.java`. The class name is `ClientThread`. Another private class `ClientJobs` allows client to listen for any incoming messages from the server, without needing to send any data to the server. This is written in `Client_2015085r1.java` file.
Server must run before the clients.

Compile the code using the following lines:
```
javac Server_2015085r1.java
javac Client_2015085r1.java
```

Run the code:
```
java Server_2015085r1 <port>
java Client_2015085r1 <host> <port>
```

`ClientThread` class implements most of the additional functionality (needed for this assignment). First, the class decodes the type of message the sender client wants to send. It can be of 3 types: `Client X,Y,Z: Msg`, `Server: List All` and `All: Msg`.
Any input other than the ones specified above is treated as invalid, special care needs to be taken about the formatting.

Following are the ways in which the server handles these:

1. `Client X,Y,Z: Msg`

   It decodes the target clients (`X,Y,Z`) and uses their `OutputStream`s to send the message received from the sender client. In case the client `Y` does not exists or `Y` is an invalid number (e.g. 0, −1, etc.), the server generates an error saying: `The client Y does not exist`, and sends it to the sender client. Also, the formatting needs to be followed strictly or it will cause an error. Additionally, whenever a client receives a message, we show the sender client as well (e.g. `Received from Client 2: Hi`).

2. `Server: List All`

   The server maintains a hash map of all the active clients. It contains the `clientID` (key) mapped to the `ClientThread` (value) on which it is running. The map is sorted in ascending order based on the keys (called `TreeMap`). Whenever a client joins, it is assigned a unique `clientID` and a thread. The entry is made in the map. Whenever a client exists, the thread is stopped and the entry removed. In order to list all the active clients, the server goes through all the keys listed in the map and sends them to the client in the form of a comma separated list.

3. `All: Msg`

   Here, we again make use of the hash map to find the active clients and send the message to all of them.

Special case: Suppose we have 3 active clients – 1, 2, 3. `Client 1` exits but later connects back. `Client 2` wants to send a message to `Client 1`. It cannot do so as `Client 1` does not exist anymore. Instead, when it rejoined, it was assigned a `clientID` of 4. So, `Client 1` is `Client 4` now. And we cannot know whether the client that left and rejoined, is an old client or a new client, i.e, we cannot know that `Client 1` is `Client 4`. We treat every client as a *new* client. So, `Client 2` can do nothing but send the message to `Client 1` and receive an error. He can always see the list of active clients by asking it from the server.