Chat server (Manuel Furia)

1) Modifications to the specification of the basic project

The format of the message has been modified to be more versatile towards more advanced features.

A message can be sent by the client without additional information, in which case it will be addressed to the main room of the server. To specify another room instead, a message will be preceded by @name_of_the_room.

Example:

```
hello everybody!      (will go to the main room)
@box hello room!      (will go to the room “box”, if it exists and the user has joined it)
```

To create a new room, or join an existing room, use the command :room room_name.

Also commands are addressed to rooms. For example:

```
:users            (will show all the users in the main room)
@box :users       (will show all the users in the room “box”, if it exists the user has joined it)
```

The admin can issue the command “:users all” to show all the users in all the rooms.

Server responses to the clients are preceded by “:-”, and clients can’t start messages with the character sequence “:-”.

The command “:messages” will produce all the messages of the room to which it’s addressed, or the main room in case no room is speficied:

```
:messages    (shows all the messages in the main room (hall) in readable format)
```

Example:

```
:messages
:- @hall     [16-09-2018 23:40:47.762] aaa: hello
```

Meanwhile:

```
@box :messages  (shows all the messages in the room box in a readable format)
```

To get an easy to parse format(@room$timestamp+user message), use “:messages data”

Example of output:

```
:- @hall$1537129799713+aaa hi
```

The admin can issue the commands “:messages all” and “:messages data all” to obtain messages from all the rooms (sorted by increasing time).

The command “:user” and “:quit” work like in the original specification.

NOTE: TopChatter has been limited to automatically showing only the number of messages written by a user when they leave (long automatic messages are annoying). To show the Top 4 chatter users, any user can just write “!topchatter” (note the exclamation mark), in a normal message in the server’s main room.

2) The structure of the program

The program is composed of a series of immutable and two main mutable components.
The mutable components are **ChatServerListener** and client handlers inheriting from the **ThreadedClient** abstract class (namely **TCPClientHandler**, **ServerConsole** and **TopChatter**). The observer pattern is used to notify events from server to client handler and from client handler to server. **ChatServerListener** is an Observable<ChatServerEvent> and an Observer<ChatClientEvent>, while **ThreadedClient** is an Observable<ChatClientEvent> and an Observer<ChatServerEvent>. In this way, the server observes the clients for client events and the clients observes the server for server events. An additional interface has been derived from the Observable, the SelectiveObservable<ChatServerEvent>, to allow the server to send a message to a specific client when necessary (for rooms and private messages).

The server state (ChatServerState) is immutable, and contains users, rooms, message history and other useful information. The server state is updated by generating a new one. Output actions to be taken (like sending messages, dropping clients, etc…) are appended to a list of ServerOutput objects that will be later executed by **ChatServerListener.**
ChatServerListener holds a reference to the current server state, takes care of updating it and executing the list of actions produced while constructing the current state.

The server commands are all stored in a Map String → anonymous functions, with the anonymous functions taking parameters containing the immutable server state, and producing a new one. To add a new command is enough to append it in the Map in the **Commands.kt** file. The **Interpreter** class takes care to parse the text message and fetch the appropriate anonymous function from the Map.

The program also contains one singleton, **Constants,** holding only immutable constants used throughout the program.

3) Additional features

The server supports the following commands:
:user, :users, :messages, :quit, :admin, :room, :leave, :kick, :grant, :KICK, :BAN, :UNBAN, :topic, :query, :rooms, :whitelist, :blacklist, :pvt, :block, :unblock, :ping, :STOP, :schedule, :execute.
These commands are documented in Commands.kt.

The server will timeout clients that don’t do any action or ping (by writing the special hidden message “:PING:”) within a certain amount of time (set to 10 minutes by default, see Constants.kt).

The server will load plugins placed in the specified folder at startup, as kotlin scripts, adding all the additional commands dynamically to the command Map. An example of plugin can be found in the directory “plugins”. Because of this feature, the kotlin-compiler, kotlin-reflect, kotlin-script-runtime and kotlin-script-utils libraries are added as dependencies to the project.