UNIVERSITY *of* NEW ORLEANS

DEPARTMENT OF COMPUTER SCIENCE

CSCI 4311/5311 Computer Networks and Telecommunications

Fall 2017

Programming Assignment #2 (PA2): `msgp chat`

Assigned: Monday, October 9, 2017

Due: Sunday, October 29@11:59pm

Objective:

The purpose of this project is to implement a simple chat application. The required implementation consists of client and server components, which communicate via the *MsgP* protocol (specified below).

The basic functionality of the server is as follows:

- It maintains a non-persistent list of chat groups (groups are created on demand and are not persistent across server executions).
- It allows clients to join any of the existing groups, or create a new one implicitly by requesting to join a group that does not exist
- Upon request, it provides to clients a list of the available groups, the current membership of the group, and the history of message sent to a particular group.
- It allows clients to send messages to individual users, a specific group, or multiple groups.

MsgP Spec:

All commands are in plaintext ASCII format, and they all start with the string **msgp**; users and groups are alphanumeric strings of up to 32 characters.

Join → Adds a user to a chat group; if the group does not yet exist, it is created (reply code 200); if the user is already a member, no action is required (reply 201). Format:

```
msgp join <user> <group>
```

Example:

```
msgp join alice 4311
```

Leave → Removes a user from a chat group (200); if user is not a member (but the group exists) return 201. If the group does not exist, return 400. Format:

```
msgp leave <user> <group>
```

Example:

```
msgp leave alice 4311
```

Groups → Enumerated the current list of groups, including empty ones (200); return 201, if empty result. Group names are returned one per line, with a blank line marking the end of the list. Format:

```
msgp groups
```

Example response:

```
msgp 200 ok
csci
4311
5311
<cr><lf>
```

Users → Enumerates the current list of users for a given groups (200); if empty result, 201; 400 if no such group exists. User names are returned one per line, with a blank line marking the end of the list. Format: Format:

```
msgp users <group>
```

Example:

```
msgp users 4311
```

Example response

```
msgp 200 ok
alice
bob
<cr><lf>
```

Send → Sends a message to the given list of recipients; group messages becomes part of the history for that group. The sender is given by a `from:` header clause, the recipients by one, or more, to: headers. User names are prefaced with the @ character, whereas groups by the # character. Returns 200 upon success, 400 if any of the recipients does not exist. Format:

```
msgp send
from: <user>
to: @<user>|#<group>
to: …
<cr><lf>
<single line message>
<cr><lf>
```

Example:

```
msgp send
from: alice
to: @bob
to: #4311
<cr><lf>
hi bob!
<cr><lf>
```

History → Provides a chronological list of messages for a given group (200); 201 if empty result, 400 if no such group. Format:

```
msgp history <group>
```

The format of the messages is *exactly* the same as in the case of send. For example, the request:

```
msgp history 4311
```

Would yield something like:

```
msgp 200 OK
msgp send
from: alice
to: #4311
<cr><lf>
hello everyone!
<cr><lf>
msgp send
from: bob
to: #4311
<cr><lf>
sorry for being late!
<cr><lf>
```

Reply codes → Server always precedes its answer (if any) with one of three reply code, single-line messages:

```
msgp 200 OK
msgp 201 No result
msgp 400 Error
```

User Agent

The agent must show a prompt of the form **`@[user]>>`**, e.g., **`@alice >>`**.

Messages received must be displayed in the form

```
[<sender>] <message>
```

e.g.

```
[bob] sorry I am late
```

Your user agent must read all user input from the standard input stream, and must support the following single-line commands:

- `join <group>`
  Joins the specified group and reports current number of members. Example:

  ```
  join 4311
  Joined #4311 with 3 current members.
  ```

- `leave <group>`
  Leaves the specified group, no output on success. Returns error if trying leave a group of which the user is not a member:
  ```
  leave 4311
  ```

```
leave 4311
Not a member of #4311.
```

- `groups`
  List the currently available groups and the number of members; no groups → no output. Example:

  ```
  groups
  #4311 has 12 members
  #5311 has 3 members
  #test has 0 members
  ```

- `users <group>`
  Lists group membership; error if no such group, nothing of no members. Example:

  ```
  users 4311
  @alice
  @bob
  ```

- `history <group>`
  List the log of messages for a particular group. Example:

  ```
  history 4311
  [alice] hello
  [bob] sorry I am late
  [charlie] let's get started
  ```

- `send <recipient list> <message>`
  Sends a message to users & groups on the list; error if any of the recipients do not exist. Individual users are prefaced by @ and groups by #. Messages sent to a group(s) must become part of the history of all those group(s). Example:

  ```
  send @bob #4311 #5311 when is the assignment due?
  ```

Program invocation

Client:

```
java csci4311.chat.CLIUserAgent <user> <server> [<port>]
```

Server:

```
java csci4311.chat.ChatServer [<port>]
```

The default port (if not specified) is 4311.

Implementation requirements

- This is an individual assignment—the submitted code must yours, and yours only. (*You can use snippets of example code discussed in class.*)
- You can have as many other classes as you need but ***all*** must be members of the `csci4311.chat` package.
- Your client module must be split in two parts:

    - `csci4311.chat.CLIUserAgent`, which implements the `csci4311.chat.UserAgent` interface, and
    - `csci4311.chat.TextMsgpClient`, which implements the `csci4311.chat.MsgpClient` interface; the two must only interact via these specified interfaces.
- The `csci4311.chat.TextMsgpClient` class must ***only*** implement the *msgp* protocol specified above and have ***no*** user agent functionality.
- The `csci4311.chat.CLIUserAgent` class must ***only*** implement user interaction, and ***no*** protocol details.
- You must have two main classes `csci4311.chat.ChatServer` and `csci4311.chat.CLIUserAgent` that can be executed as specified above.
- You are only allowed to use classes that are part of the standard JDK (`java.*` packages).

Submission

- ***No late submissions***. You have three weeks, which is significantly more time than you need, so that you can plan your schedule and accommodate various commitments.
- Name your submission files as required above (along with any other files you need). ***Only submit source code*** and any documentation you feel is necessary.
- `git push` it to your private repo on `gitlab.cs.uno.edu:<your-login>/csci4311-f17` and that the instructor (`jtsylve`) must have *reporter* access.

Note: Make sure your repo is named ***exactly*** as specified in the convention above. At the end of the semester, the repos will be garbage collected (you will be given notice to archive your work).

- Place your submission in a `/pa2` folder.
- Your commit message must read "`submission: pa2`".

Scoring:

This PA is worth 75 points

Extra Credit (*mandatory* for 5311 students) +20%

- Define the `MsgpServer`, and `MessageServer` interfaces and organize your code such that there is a clear separation of concern between the communication protocol and the core chat server functionality.
- The class implementing `MsgpServer`—`TextMsgpServer`—should only be concerned with the sending and receipt of correcting encoded messages (just like the `TextMsgpClient` class).
- The class implementing `MessageServer`—`ChatServer`—should only be concerned with the core functionality of maintaining users, groups, messages, etc. It should only operate in terms of *Java* classes and must be completely oblivious to the implementation of the protocol (just like the `CLIUserAgent`).

You should only attempt extra credit ***after*** *you have successfully completed the primary assignment.*