Chat Channels with Grading Notes

In this assignment you will create a simple client/server tool which implements a simple communication protocol. The server in this case will enable clients to send messages between other clients using a *topic based* publish/subscribe system.

1 The Communication Protocol

The client and server will speak a protocol built on the following message types:

`REGISTER topic` The line begins with REGISTER (capitalization isn't important) followed by one space. It's followed by a single word which is the topic of the message. Capitalization of the topic **is not** important. The line ends with a new line character.

`LEAVE topic` The line begins with LEAVE (capitalization isn't important) followed by one space. It's followed by a single word which is the topic of the message. Capitalization of the topic **is not** important. The line ends with a new line character.

`SEND topic sentence` The line begins with SEND (capitalization isn't important) followed by one space. It's followed by a single word which is the topic of the message. Capitalization of the topic **is not** important. After the topic comes one space. After the second space comes a sentence which may contain spaces. The line ends with a new line character.

`FORWARD topic sender sentence` The line begins with FORWARD (capitalization isn't important) followed by one space. It's followed by a single word which is the topic of the message. Capitalization of the topic **is not** important. After the space comes a sender IP and port string in the form 0.0.0.0:0000 (*ex.* 127.0.0.1:5005 or 10.0.251.17:4295) which is an IP address followed by a colon (:) and a port number. After the sender IP and port string comes a single space. After the space comes a sentence which may contain spaces. The line ends with a new line character.

Note: The `sender` field contains the IP address and port number of the **original sender of the sentence**, **not the server** which is forwarding the sentence.

`OK` The line contains just the letters OK, but capitalization and extra trailing spaces are not important. The lines ends with a new line character.

`ERROR` The line contains just the letters ERROR, but capitalization and extra trailing spaces are not important. The lines ends with a new line character.

`CLOSE` The line contains just the letters CLOSE, but capitalization and extra trailing spaces are not important. The line ends with a new line character.

The protocol messages above are the ones sent by the client and server programs. The user will not need to write the commands above or see the responses in the protocol. Instead, the user will interact with a client interface program as described in the following section.

2 The Client Program

When the client program starts, it first reads from a configuration file which contains the following information: (1) a server IP and (b) a server port. It stores the server IP and server port for later, when the user requests to connect. This will let the user point the client program to a different server without needing to recompile the code and without the user needing to enter the server IP and port manually. The configuration file will be read when the client program starts up and doesn't need to be read again during execution.

Note: You choose how to implement the configuration file, but you must document the configuration file and how it works in the README file for the assignment.

The client program lets the user perform the following operations:

Connect Connect to the server listed in the configuration file. The user can then perform any number of the operations below during a single session.

Once connected, the client will begin listening for messages from the server on a dedicated listening thread. It will show messages that it receives (`FORWARD topic sender sentence`) on the screen in the following format:

(topic) senderIP:senderPort hour:minutes:seconds - sentence

For instance, the following are some messages that might be shown:

(horses) 127.0.0.1:5965 12:10:45 - I like horses!
(horses) 127.0.0.1:4857 12:10:50 - So do I!
(cats) 10.0.251.17 12:11:15 - Meow

Disconnect If the client program is currently connected to the server, it disconnects by sending a `CLOSE` message to the server. The user can later connect again to the server without needing to restart the client program.

Once the user has chosen to disconnect, it must stop receiving forwarded messages from the server.

Quit Close the client program. If the client is currently connected, it must perform the steps for "Disconnect" above to disconnect from the server and then quit.

Register for a topic If the client program is connected, the user can enter a topic to register for. The client program then sends a `REGISTER topic` message to the server and receives either an `OK` or an `ERROR` message in return. If the user is registered for the topic, an *error* message is shown to the user (the server returns `ERROR` if the client tries to register twice for a message). Otherwise, a *success* message is shown to the user (the server returns `OK`).

If the client isn't connected to the server, the user can't register. The client program may optionally show the user an error message.

Leave a topic If the client program is connected, the user can enter a topic to leave. The client program then sends a `LEAVE topic` message to the server and receives either an `OK` or an `ERROR` message in return. If the user isn't registered for the topic, an *error* message is shown to the user (the server returns `ERROR` if the client tries to leave a topic it's not registered for). Otherwise, a *success* message is shown to the user (the server returns `OK`).

If the client isn't connected to the server, the user can't leave. The client program may optionally show the user an error message.

Send a message If the client program is connected, the user can send a new sentence to the server with a topic provided by the user. The user does not need to be registered to the topic to send a message under it. The client sends a message of the format `SEND topic sentence` where `topic` is the topic that the user chose and `sentence` is the sentence that the user entered.

If the client isn't connected to the server, the user can't send a message. The client program may optionally show the user an error message.

As noted above, the sending a message to the server will cause the server to forward the message to all clients registered for the topic.

Additional Client Requirements

1. The client program must offer either a GUI (Java Swing or JavaFX) or text based (command line) interface with the above operations available.
2. The user must not be required to enter raw protocol commands such as REGISTER, LEAVE, or SEND.
3. Protocol messages must be sent by the client program in response to user requested operations usingthe GUI or text based interface.
4. The choice of the interface and its design is up to you.
5. You must document the user interface and how it works in the README file for the assignment.

3 The Server Program

The server program offers a multi-threaded interface to the topic based chat service. When the server starts up, it first reads from a configuration file which has *the server port*. The server loads the information from the file when it starts up and keeps it ready for later use.

Note: You may choose how to implement the configuration file, even making more than one server configuration file or adding more data to it if it's convenient. You must document the configuration file and how it works in the README file for the assignment.

The server program lets the user perform the following operations:

Start Listening The server automatically shows the user the server computer's current IP addresses and allows the user to either:

- Choose one from the list **or**
- Listen on all available addresses (0.0.0.0) **or**
- Listen on an IP address not in the list

The server program can implement this either

(a) Graphically by displaying a list which contains the computer's IP addresses and enables editing **or**
(b) Using the command line by showing the current IP addresses and allowing the user to eitherchoose one of the options above before starting to listen.

Once the user has approved the server IP, the server program starts to listen on the IP address provided and the server port retrieved from the configuration file.

Once listening, the server program must handle multiple concurrent conversations with clients. Each concurrent conversation must be handled by a separate worker thread (*ex.* Thread). Assume the number and rate of connections will not be large enough to require the use of a thread pool (but you may use a thread pool if you wish).

Stop Listening If the server is currently listening, it must stop doing so. All worker threads must be closed down as soon as possible. Once the user has told the server to stop listening, it must stop forwarding messages to clients.

The user can choose to start listening again without needing to restart the server program.

Quit If the server is currently listening, it first does the actions listed above in "Stop Listening." It then exits the program.

Additional Server Requirements

1. The server must offer either a GUI (Java Swing or JavaFX) or a text based (command line) interface.
2. The choice of the interface and its design is up to you.

3. You must document the user interface and how it works in the README file mentioned below.

3.1 Worker Threads: Handling a Client

Each worker thread for the server program must be able to handle a single conversation from a client program. The worker thread must perform the following actions in response to messages from the client:

`REGISTER topic` The worker thread checks if the client is currently registered to the topic listed. If **yes**, it replies `ERROR`. **Otherwise**, it responds `OK` and adds the client to the list of clients to which messages marked with the topic are forwarded. Once *registered* to the topic, any future messages sent to the topic will be *forwarded* to the client. Capitalization does not matter for topic names.

`LEAVE topic` The worker thread checks if the client is currently registered to the topic listed. If **not**, it replies `ERROR`. **Otherwise**, it responds `OK` and removes the client from the list the of clients to which messages marked with the topic are forwarded. Once *removed* from the topic, any future messages sent to the topic will *not be forwarded* to the client.

Capitalization does not matter for topics.

`SEND topic sentence` The worker thread replies `OK`. It then checks which clients are registered for `topic`. For each registered client, the worker thread causes a `FORWARD` message to be sent in the format: `FORWARD topic sender sentence`. In the FORWARD message, the fields are determined as follows:

- The value `topic` is the same as was in the SEND message.
- The value `sender` is the IP address and port (*ex.* 10.0.251:1050) of the client who originally sent the SEND message.
- The value `sentence` is the sentence that was in the SEND message.

If there are no clients registered to the topic, no forwarding is done.

`CLOSE` The worker thread closes the connection. There is no reply.

Other If any other message or an illegally formatted message is received from the client program, the worker thread replies `ERROR`.

3.2 Log

The worker threads and server program must keep a verbose log of each operation. The log may be printed to the console, shown in a text box, or stored in a dedicated log file. If you choose to use a log file, its location must be documented in the README file for the assignment.

Server (General) Each log line from the server must include the date and time. The server log output must contain the following information at least:

- An opening message indicating that listening has started on a port and IP address.
- A closing message indicating that listening has stopped.
- Any errors or unexpected exceptions which arise during the course of the server's operation.

Worker Threads Each log line from the worker thread must include the date, time, client IP and client port. The worker thread log output must contain the following information at least:

- A "connected" message for every new client connection (with IP and port of the client program)
- A "disconnected" message for every client connection which ended (with IP and port of the client program)

- An echo of each complete command or action which occurred or which the client program sent (register, leave, send, unknown).
- An echo of each reply the worker thread sent back (ok, error, forward).
- Any errors or unexpected exceptions which arise during the course of the server's operation.

3.3 Thread Safety

Since there may be more than one worker thread running at once and each one may try to register to topics, leave topics, and send messages at the same time, you will need to implement a *mutual exclusion* mechanism to prevent state problems in the topic and message queues. Read the Java documentation to see which classes are *thread safe* and learn about the `synchronized` key word and the `Lock` interface in Java. There are a few ways to achieve thread safety, some easier and some more complicated. You may use any one of the ideas below in your code or you may come up with your own idea:

1. Use `synchronized` on all critical blocks or methods.
2. Implement a topics and messages class which enforces thread safety by using *synchronized* where needed.
3. Use an SQL database to store the topic registration and messages to send. (**Note:** If you choose this option you must ensure the database can be submitted with the assignment and accessed on any Windows 7 or higher computer).
4. Use a `ReentrantLock` or `ReentrantReadWriteLock` object to synchronize access to the list of registered clients and messages.

Document your thread safety strategy in the README file for the assignment.

4 Additional User Interface Requirements

Both the client and server programs must support the following requirements:

1. Language: The client and server programs must be written in Java.
2. Robustness: The server and client programs must perform error handling, not crashing due to unhandled exceptions.
3. Responsiveness: The server and client programs must enable the user to interact with the user interface when listening or processing. This implies that network communications must not be done on the same thread as the user interface.