Exercise 2

Concurrent Programming

1 Goal

The goal of this exercise is to teach (remind) you how to program threads in Java. To that end you will implement a local chat application. In this application, different users will be able to send messages to each other communicating via a local shared memory.

2 Guidelines

We present here in more details the different indications that will help you to implement the local chat application.

The application will simulate a distributed chat but in a local environment. To that end, the main chat application will launch several chat client windows. Each of them will allow a user to connect/disconnect, and send chat messages to the other users. The messages will be sent, via method calls, to a server object which will be responsible for storing the connected clients and dispatching the messages to the clients. Please note that all the clients communicate together.

2.1 Application Overview

The user should be able to launch the application via the following command:

```
java CentralizedChat <nbOfClients>
```

And `nbOfClients` windows like the one presented in Figure 1 should appear.

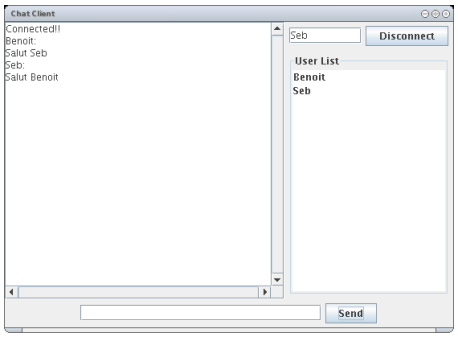

Figure 1: A Chat Client GUI

A user is able to connect/disconnect (via the `Connect` button), send messages (via the `Send` button) and receive messages as well as the list of the connected users. Your application does not have to be exactly the same as presented in Figure 1, but the same functionalities should be provided.

2.2 General Architecture

The chat application should follow the architecture presented in Figure 2 (please note that the interfaces `IChatClient` and `IChatServer` of Exercise 1 are not depicted in Figure 2).

In short, different `ChatClient` instances, communicate via a `ChatServer` and display the information to a `ChatGUI`. In order to ease you the work for Exercise 3, try to design your

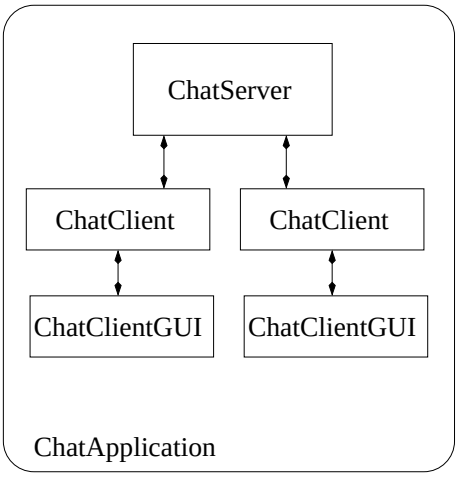

Figure 2: Sketched Chat Architecture

application the most generic as possible (i.e., in using Java interfaces between the different actors). Finally, we require that your implementation defines and uses its own blocking queue to manage the list of messages; we explain what a blocking queue is in Section 2.4.

2.3 Threads

As several clients will be able to run simultaneously in the same main process, you will have to use `Thread`'s. Those threads will call the same methods on the server, hence you will have to use the `synchronized` keyword. Please have a look at the course or at the Java Documentation in order to know how to deal with this concept.

2.4 Synchronization

Threads in the application synchronize together using one or several blocking queues. A blocking queue uses an instances of class `ArrayList` in its core. As an instance of class `ArrayList` is not multithread-safe, you need therefore to add the necessary synchronization code, for using a blocking queue within our multithreaded environment. The specification of the blocking queue is as follows:

- `get`: this method pops the first message in the queue and blocks if there is no such message.
- `put`: this method pushes a message at the end of the queue and notifies threads waiting for new messages.

Be careful to avoid deadlocks of threads.

3 Due

You have to give, for April 11th, the complete source code of your application. You can send your code, **without** the compiled classes, in a **.zip** file archive (or **.tgz**) to the following email address: *Sebastien.Baehni@epfl.ch*. The archive **has** to contain a script file used for compiling your code (either on Windows or on Linux). Your email should arrive before noon and its subject has to be: *IDS EX2*. Please put also in the content of your mail the firstname and name of your partner.

The application has to run correctly (i.e, without bugs) for you to receive 0.33 bonus points. If the code is well designed, well written and generic, you will received additionally 0.33 points. Finally, if the code is well documented (i.e., javadoc preferably) you will get the final 0.33 points of the bonus.

References

[1] Java API Documentation. *http://java.sun.com/j2se/1.5.0/docs/api*.