CM102228 Coursework 1
Simple Chatting System

February 11, 2019

1 Introduction

In your first programming assignment for this semester you will create a simple chat system. You will have to code both a *client* and a *server* component for the system.

The deadline for submission is Friday 15th March 2019 @ 8pm.

2 Specifications

2.1 Server

The first component you have to code is a *multi-threaded server*. You will have to spawn threads to handle incoming requests in parallel. In addition to the list of requirements below, we will mark your approach on networking and concurrency. Make sure you use locks or implement concurrent data structures, as required, to make the chat server thread-safe.

1. The server can handle connections by multiple clients.
2. The server accepts clients connecting through a TCP/IP connection, using Java's *Socket* API.
3. The server can receive messages from clients.
4. The server broadcasts all messages received to all connected clients.
5. The server continues running if one or more clients disconnect from it.
6. The server shuts down cleanly if its user enters *EXIT* on the terminal.
7. The server uses 14001 as a default port.
8. The server's implementation contains a class called *ChatServer*, allowing the software to start by running the following on *linux.bath*:

   ```
   java ChatServer
   ```

9. You can pass an optional parameter, called *csp*, to ChatServer to request that it bind to another port. For example:

   ```
   java ChatServer -csp 14005
   ```

2.2 Client

The second component you are to create is a *client*. The client is responsible for sending and receiving messages from the server. Think of it as any Instant Messaging (IM) application, for example *WhatsApp* or *MSN Messenger*. The following specifications are the minimum requirements for your client:

1. Your client must be able to read messages from the console.
2. The client can send messages to a *server*.
3. The client must be able to receive messages sent from the server.
4. The client should output all messages received by the server.
5. The client connects to the server through a TCP/IP connection, established using Java's *Socket* API.
6. The client sends and receives any messages through the socket connection established in the step above.
7. The client's implementation contains a class called *ChatClient*, allowing the execution of the software by running the following on *linux.bath*:

   ```
   java ChatClient
   ```

8. The client should use *localhost* as a default IP address to try to connect to.
9. The client should use 14001 as a default port.
10. You can pass an optional parameter, called *cca*, to ChatClient to request that it bind to another IP address. For example:

    ```
    java ChatClient -cca 192.168.10.250
    ```

11. You can pass an optional parameter, called *ccp*, to ChatClient to request that it bind to another port. For example:

    ```
    java ChatClient -ccp 14005
    ```

12. You can pass both the IP address and port of the server. For example:

    ```
    java ChatClient -cca 192.168.10.250 -ccp 14005
    ```

If your client supports reading input from the console and displaying messages at the same time, by using a multi-threaded solution, you will be awarded additional marks.

2.3 GUI

For additional marks, you may implement the advanced Graphical User Interface feature for either/both client and server. The GUIs must have the same functionality as the command line interface versions, such as specifying the IP address and port, receiving messages, and the ability to disconnect/exit.

We must be able to choose between running a command line client/server and a GUI.

3 Marking Scheme

3.1 Marks Distribution

A total of 50% can be achieved by providing working code that satisfies all the requirements set on this document. For the rest 35% you will be judged on the quality of your code. With the remaining 15% being allocated to the advanced GUI feature. The table below outlines the high-level distribution of marks.

Criteria	Max Score	Description
Client Specifications - 20		
Basic Client	10	Client satisfies core specifications.
Multi-threaded Client	10	Client can read/write from console simultaneously.
Server Specifications - 30		
Basic Server	15	Server allows at least one client to be connected.
Concurrency	15	Server allows more than one client to be connected and uses thread safety techniques.
Code Quality - 35		
Good-code practices	20	Object-oriented programming techniques, including but not limited to encapsulation and SRP, have been used. Code is sensible, with optimisation taken into account.
Formatting & Comments	15	Indentation, whitespace, sensible comments throughout the submission
Client and Server GUI - 15		
Client GUI	10	Allows input from the user, displays chat messages and feedback from the server, allows user to specify the IP address and port of server and allows the user to disconnect /exit cleanly
Server GUI	5	Displays all chat messages, a history of connect and disconnects from clients, allows the user to specify the port the server should listen on and allows the user to exit (i.e. close down the server) cleanly

3.2 Marking Guidelines

3.2.1 Threshold for Pass (40%):

1. Basic networking functions: able to demo on linux.bath connecting one user client to server.
2. Code is good i.e. well formatted and commented.

3.2.2 Good Pass (~60%):

1. Meets all the criteria of the pass threshold above
2. Provides networking support for multiple clients.
3. Code follows clean-code practices, i.e. very good use of OO design etc.

3.2.3 Distinction (70% - 85%):

1. Meets all the criteria of a good pass
2. The client is also multi-threaded
3. Concurrency principles, such as threads safety and atomic access, were taken into account

3.2.4 Distinction (85%+):

1. Meets all the criteria of a distinction
2. Implements a Graphical User Interface (GUI) for the Client
3. Implements a GUI for the Server
4. NOTE: GUI must be runnable on linux.bath i.e. use either Java Swing or JavaFX

4 Submission

By the date/time specified above, you should upload a zip file. The name of the zip file must be in the form:

CW1-yourid.zip
e.g. CW1-zl221.zip

The zip file must contain your source code and any resources files needed. You should not include packages or other subfolders. Your solution must not have any package declarations.

Failure to follow the submission specifications, by providing non-needed files or not following the .zip structure specified, may result in penalised marks.