COMP90015: Distributed Systems - Assignment 1
Multi-threaded Dictionary Server

Name: Pei-Yun Sun
Student ID: 667816
Email: peiyuns@student.unimelb.edu.au
Tutor: Alisha Aneja

Name: Pei-YunSun
Student ID: 667816

Multi-threaded Dictionary Server

1. Introduction

The multi-threaded dictionary server is a server which allows the clients to search, add, or delete words at the dictionary server concurrently. The aim of this report is to discuss the architecture, functions, error handling, and some other creativities of the system.

2. Architecture

This systems follow a thread-per-connection client-server architecture, all the communication between clients and the server are using TCP sockets, and the system is using a classic two-tier model since the server does not have to act as a client and any request information from the other servers.

The thread-per-connection architecture allows the clients to make multiple requests over the connection, so the clients will not be required to connect to the server for sending every single request.

Using TCP sockets is reliable, and it guarantees the order of the arrivals of the request, and therefore the order of the response when searching words will be correct, which provides a better user experience.

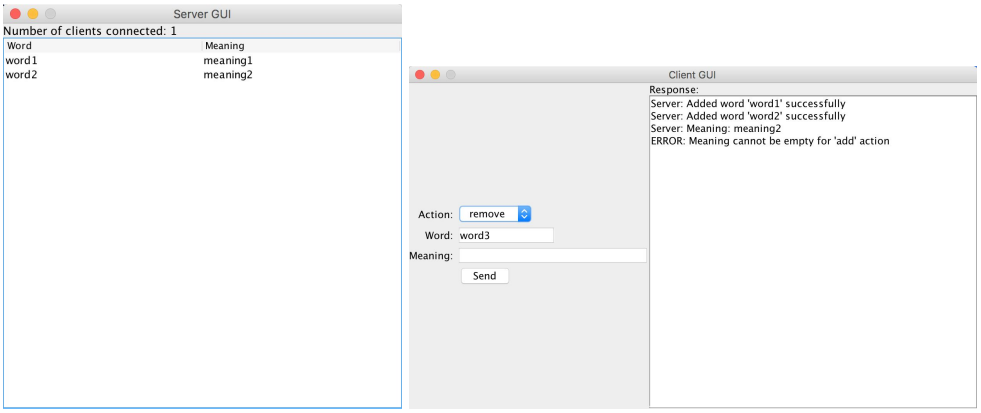
2.1. System Components

This system contains 4 classes: *Client, ClientFrame, Server, ServerFrame*. The *ClientFrame* and *ServerFrame* both extends the *JFrame* class, which are used for the Graphical User Interface (GUI).

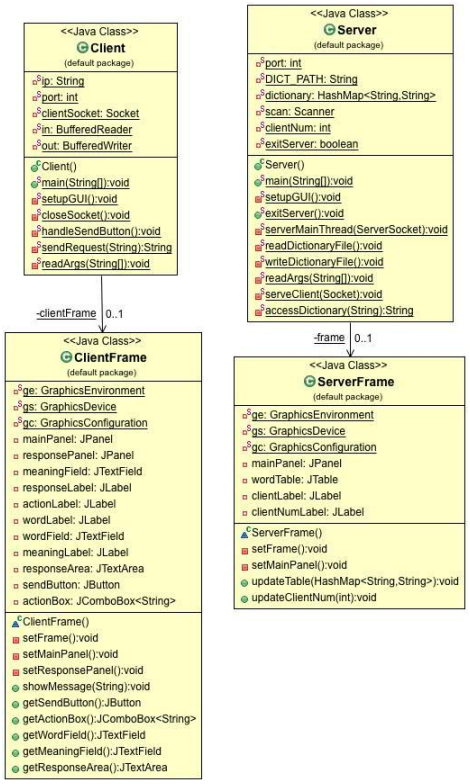
The Client and Server are the two main classes of this system, there will be one single server handling the requests from multiple clients concurrently.

The Server GUI shows the number of clients currently connected to the server, and a table of words and their meaning. When any new client connects to the server, or the existing client disconnects to the server, the number of clients will be updated. When any client add or delete a word of the dictionary, the table in the dictionary will be updated.

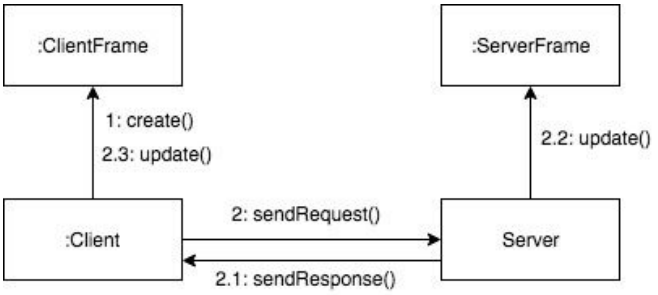
The client GUI allows the client to perform 3 types of actions: *search, add, delete*. For adding new words, the client GUI prevent the user to send the request without entering the meaning of the word. Any response from the server will be shown in this GUI.



2.1. Class Diagram



2.1. Interaction Diagram



3. Functions

This system allows the clients to *search, add,* or *delete* a word at the dictionary server, and all the changes made by a client will be visible to all the other clients, i.e. the clients are sharing a single dictionary at the server.

The clients and server communicate to each other by passing String messages. The messages sent by clients are developed by the client GUI, the GUI converts all the information about the request into a single String: "*actionType, word, meaning*", the *meaning* is optional if the action type is not "add".

The server handle the request from clients by accessing the dictionary with HashMap type (using the *word* as key, and *meaning* as value). Since there are multiple threads to serve the clients, the access to the HashMap dictionary should be synchronized. After handling the request, the server will send the add/delete successful message, meaning of word, or error message if trying the add existing word or delete the non-existing word.

4. Exception Handling

This system can handle various kinds of error including incorrect type or number of arguments, unknown ip address, sending request to disconnected server, etc.

4.1. Connecting to server

If the server or client entered incorrect number of arguments or the types of the arguments are incorrect, the exception will be caught and handled by showing the error message. The error message will tell the arguments required.

Server:

```
> java Server 1000
Invalid number of arguments (NEED: PORT & DICT_PATH)
> java Server wrong wrong
ERROR: Invalid argument types (NEED: PORT & DICT_PATH
> java Server 10000000 "dictionary.txt"
The port num is out of range, try another port num.
> java Server 3000 "dictionary.txt"
The port is already in use, try another port.
```

Client:

```
> java Client 1000
Invalid number of arguments (NEED: IP & PORT)
> java Client localhost 1000
Connection refused (no server at the port)
> java Client wrong wrong
Invalid argument types (NEED: IP & PORT)
> java Client wrong 1000
Unknown host (try other ip or port)
```

If the client failed make a connection to the server, the "UnknownHostException", "NoRouteToHostException" or "ConnectionException" will be caught, and the error message will be shown at the terminal, and if there is any error occured when getting the communication I/O streams, the exception will also be caught, and show the error message: "Failed to get I/O streams".

Any error occured when connecting to the server will present the invocation of the client GUI, and the client should to try to enter arguments with the correct format, or another IP address and port number. This also implies that if the GUI is successfully launched, the client is successfully connected to the server.

If the connection is made successfully, the number of clients connected shown in the server GUI will increment.

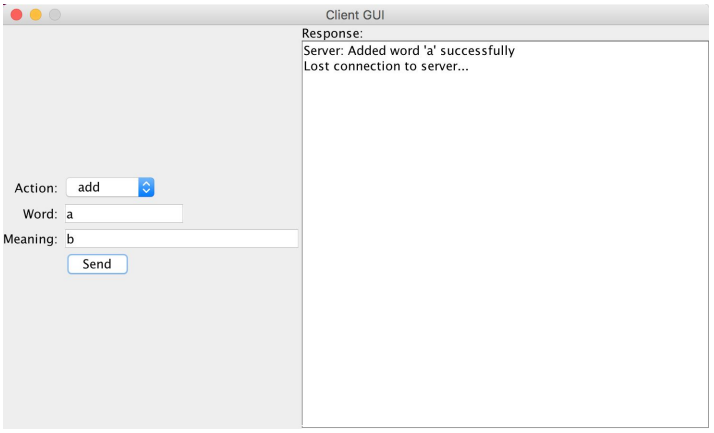
4.2. Disconnecting

If the GUI of the server or clients is closed, the application will be terminated, and before that, their sockets and I/O stream will be closed.

Closing a Server:

If the server is closed, the boolean value “exitServer” will be set to true in order to inform its child threads (for serving clients) to stop.

Afterwards, if a client tries to send request to the server, there will be an IOException, the exception will be caught, and the client will be informed that the connection is lost.



Closing a Client:

If the client is closed, the thread for serving this client of the server will close the socket & I/O streams, then stop. The number of connected clients shown in the Server GUI will also decrement.

4.3. Reading & Writing dictionary file

When the server is initialised, the dictionary will be loaded from the DICT_PATH given from the terminal argument.

Reading:

If the dictionary file does not exist, it implies that there is no prior information about the dictionary, so the program will create a new empty HashMap as the dictionary when it tries to read the file.

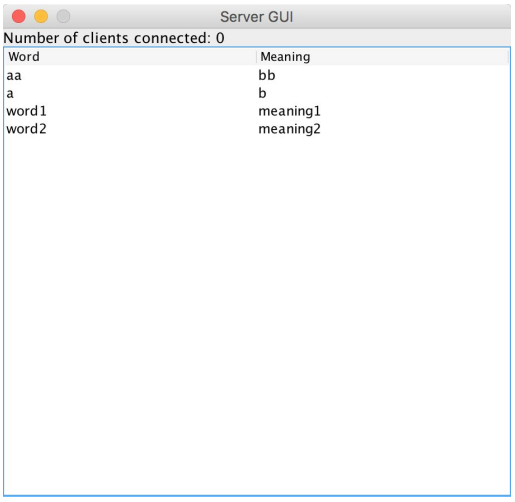
If any I/O exception occurred when reading the dictionary file, the program will use the empty HashMap as well.

Writing:

Before closing the server, the program will try to write the dictionary HashMap object to DICT_PATH, which is serializable, to the DICT_PATH. If the file does not exist, a new file will be created.

6. Creativity

Beyond the requirement of this project, a server GUI is implemented, which provides a scrollable table of word and the corresponding meaning, and there is also a label showing the number of clients currently connected to the server.



In addition, the system allows the dictionary on the disk to be updated after the server is closed, instead of reading dictionary from the disk.

7. Conclusion

The system could use the connectionless UDP sockets and thread-per-request architecture instead, which is cheaper, and it reduces the load of the server. Furthermore, the system could pass JSON Objects instead of than passing the strings, which would be more reliable and practical.