COMP90015 Assignment 1
Multi-threaded Dictionary Server

Name: HAONAN TAO
ID:1307996

1. Problem Context:

A multi-threaded dictionary system server that permits concurrent clients to access must be planned and implemented in this project using the client-server architecture.
This dictionary system provides four functions to clients: asking for the meaning(s) of a given term, adding a new word, removing an existing word, and modifying the meaning(s) of an existing word. Errors on both the server and client sides, such as I/O to and from the disc, network connectivity, and console input arguments, must be handled correctly. This project also necessitates the development of a Graphical User Interface (GUI).

2. System Components Description:

The dictionary system consists of four parts: the server, the client, the thread connections, and the GUI for the client.

2.1 Server:

Servers are implemented in the class Server. It uses TCP as a protocol that can guarantee reliability. The server handles multi-threaded clients with each connection thread. The server mainly receives the port and dictionary Path as parameters and then generates a server socket internally, and then continuously accepts the connection of the client socket through a while loop, and hands it to connections. The class that a particular method handles.
The server still uses the readDictionary()method, which reads the dictionary file from the disk into memory for further use.
Moreover, in the server part, the handles of abnormal options such as non-compliance with parameters, socket creation, IO stream, and file reading are also carried out.

2.2 Client

The client is implemented in the Class Client. It sends requests from users to the server, and handles the responses from the server, then shows them on the client GUI.
Not only open a GUI for each client start-up but also send requests to connect. (Thread per connection)
Then the getResponse() method is still used to transmit the message queue between the client and the server. The message queue here uses

LinkedBlockingQueue<String>, which fully realizes the thread safety issue of multi-thread concurrency.
LinkBlockingQueue uses the ReentrantLock keyword internally, which can be accessed by multiple threads at the same time without thread safety issues. In a multi-threaded environment, producers and consumers can operate on the queue concurrently. And provides blocking operations, such as put() and take() methods, which will automatically block the thread until there are elements or space available in the queue.

2.3 UiShow

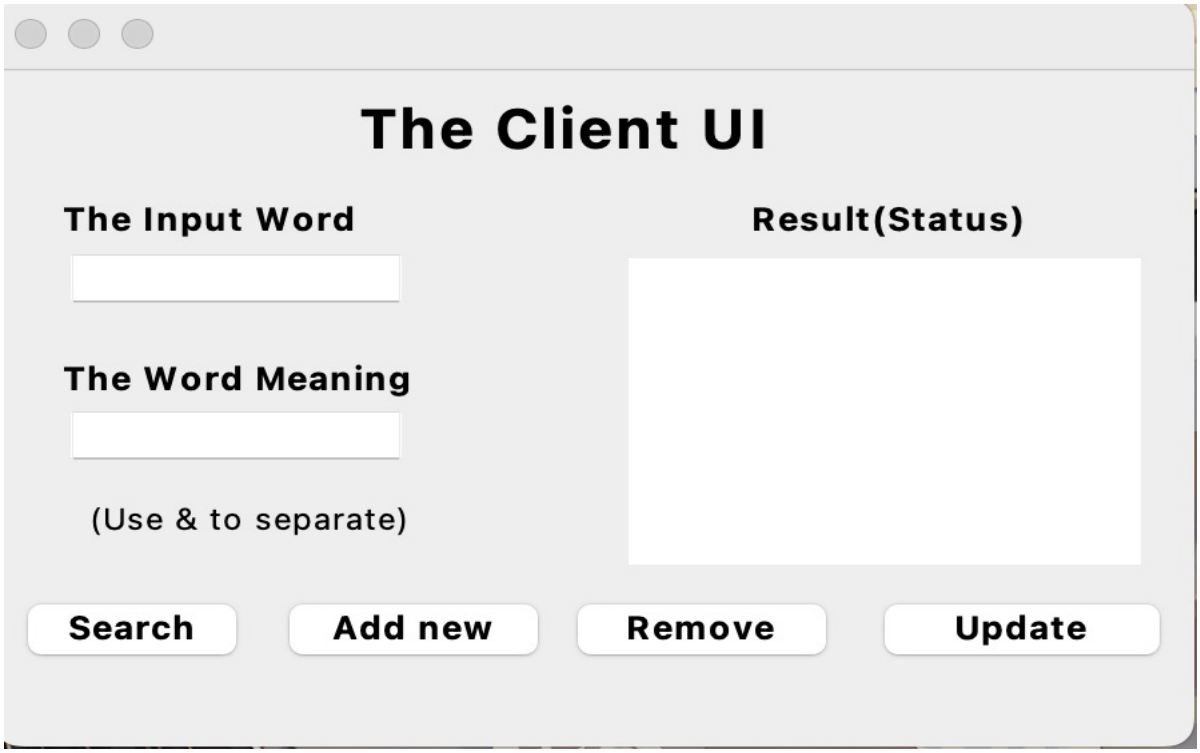

Figure. Client GUI display

Client GUI is implemented in the class UiShow. It is a GUI display interface based on javax. swing, which performs a GUI display for each client.
Four buttons (add, search, update, delete) are designed inside, and then the input boxes for words and meanings and the display boxes for results are set.
After clicking each button, call the method, use @ as a separator to divide the command into action, word, and meaning, and then passed into client-connection to execute the corresponding method.
Perform case-sensitive processing on letters, change all input words to lowercase, and avoid failure of methods with the inconsistent case.
Then, for a word with multiple meanings, use & to split multiple meanings, use \n to replace in ResultShow and then display in a new line.

2.4 Connections

Connections are implemented in class Connections.（Thread-per-connection）

Mainly, after the server receives the request from the client, it passes the socket as an input into the connection and then starts a connection thread to implement a specific client method.
It performs specific judgments and writers on the four methods: add new, search, remove, and update.
First, read the request from the UI button clicked, and then divide the string into action, word, and meaning, look up the word in the dictionary, write 0 if it can be operated, and write 5 if it cannot be processed.
Then each operation executes the writeDictionary() method to write the dictionary from memory to disk (except search), and then handle possible errors.

3. Class Design:
Designed for a single-server, multi-client architecture, the dictionary is stored in the Server part.
Then for the reliable connection of TCP, there is no need to consider packet loss and unstable connections such as UDP.
Use Socket and Thread to achieve multi-threaded concurrency, thread-per-connection to pass each client socket into the connection and do specific method processing in the connection.
Each Client opens a GUI page.
For the server and client, LinkedBlockingQueue<String> is used for message queue transmission to ensure thread safety.
Then use Concurrent HashMap<String, String> to read and write between disk and memory, also because it is a read-safety HashMap.

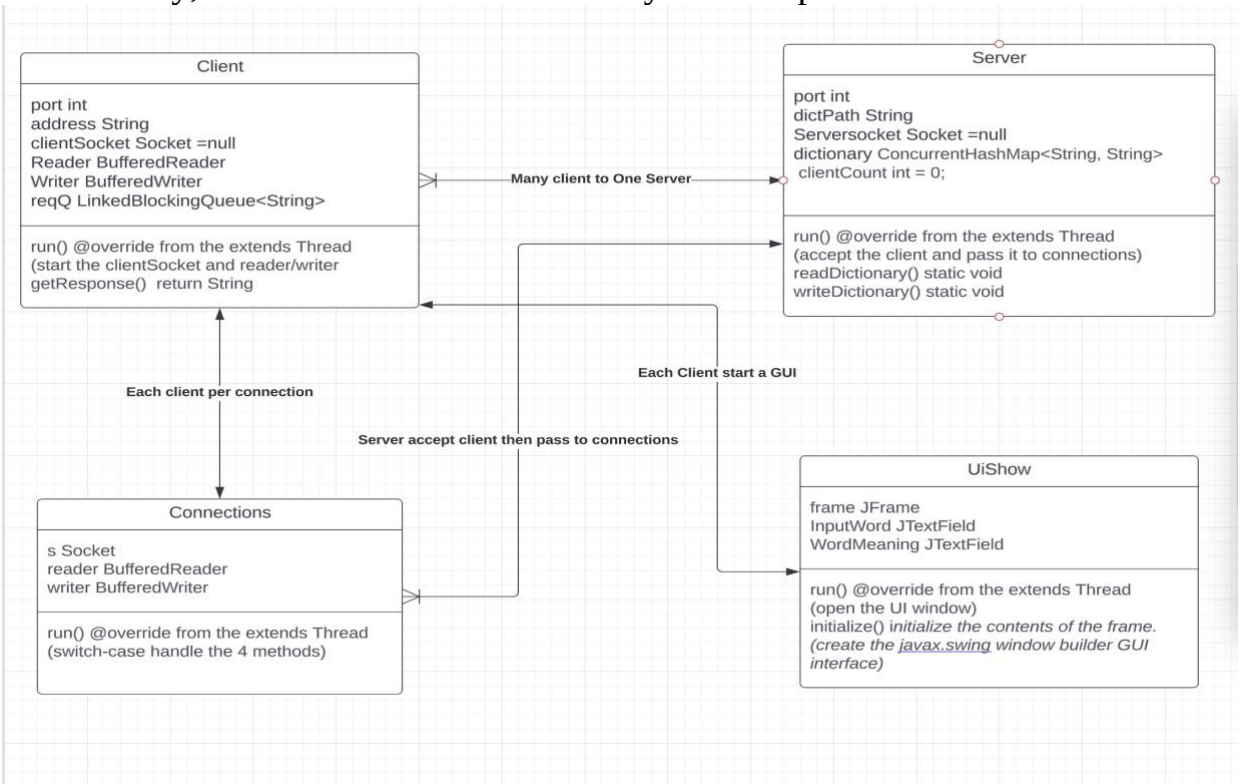

Figure. UML

4. Interaction graph:

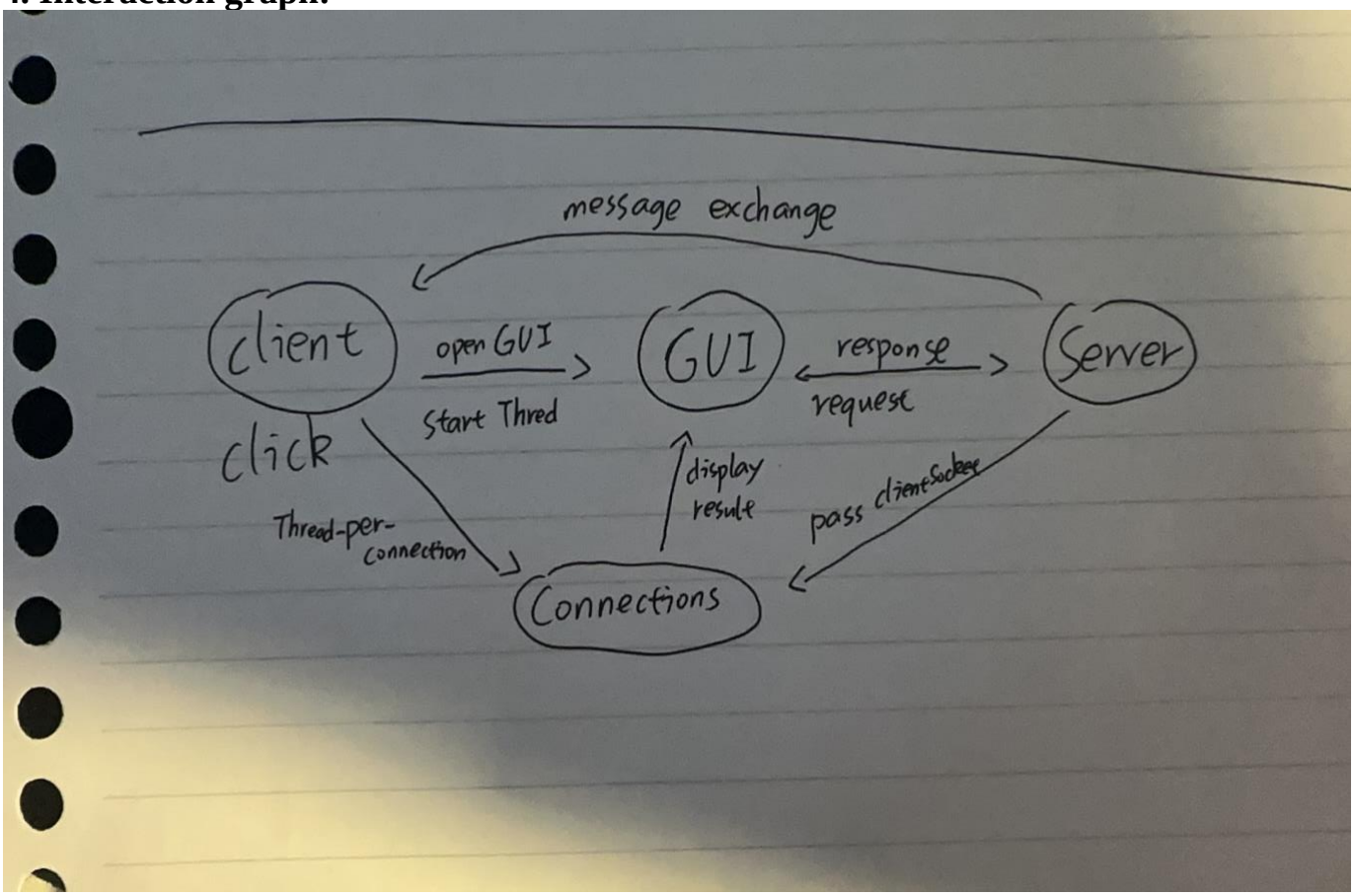

Figure.Ineractrion Step Graph

Server: For the Server, enter port and dicPath as inputs to start a server thread. Then create a server Socket, continue to accept client Sockets, and each client socket is passed to the connection (Thread-per-connection).

Client: On the client side, accept the address and port as inputs. Then each client opens a client Socket area to connect to the server and opens a GUI page.

Connections: Each client opens a connection, receives the client socket in it, writes and reads the I/O stream, and can receive the instruction from the UiShow button from the client, split it, and send it to the client after splitting the command (let the client knows whether it can continue to execute the subsequent method).

UiShow: Simple GUI interface, javax. Window builder interface for swing. Then there is the corresponding add new, update, remove, search buttons, and for each button, start the corresponding method, get action, word, meaning, use "@" to divide, write Req, and facilitate the client and server message

connection. And get the response from the server to determine how to display ResultShow.
(A variety of meanings are separated by & and displayed with \n， different lines)

5. Error Handle
I am considered different types of errors will occurs in a different class:

args.length != 2
port <= 1024 || port > 65535
address.equals("localhost")
NumberFormatException
Exception
UnknownHostException
ConnectException
IOException
SocketException
NumberFormatException
IllegalArgumentException
FileNotFoundException
UnsupportedEncodingException
NullPointerException
ArrayIndexOutOfBoundsException

6. Critical Analysis

- **Advantages**

1. Compared with UDP, the connection of TCP is more reliable, and there will be no packet loss.
2. The Thread-per-connection method can improve the concurrent processing capability and response speed of the system. Each connection has its own thread, which can process requests independently, avoiding the situation of blocking other connections due to the long request processing time of a certain connection.
3. Use a simple txt dictionary to read and write, with high efficiency, and use LinkedBlockingQueue<String> and ConcurrentHashMap<String, String> inside the project, both of which consider the multi-threaded concurrency security issue.

- **Disadvantages**

1. Thread-per-connection won't be a good choice for many connections, because the repeated creation, destruction, and switching of threads cause large overhead. It should use the thread pool for many connections.
2. Using String to communicate between the server and client can lead to some errors when splitting the String
3. Using a TCP connection may take too long, and there is no serialization or JSON format

7. Creativity elements

1. Case-sensitive considerations are made for the input of words, and all become lowercase to avoid the failure of query and other methods
2. Use Buffered Reader and Buffered Writer to increase reading and writing efficiency
3. LinkedBlockingQueue<String> is used, which uses ReentrantLock keywords and ReentrantLock locks to prevent multiple threads from executing tasks at the same time. （ReentrantLock, like synchronized, can be used to ensure thread safety in a multi-threaded environment）
4. Use the thread-safety HashMap of ConcurrentHashMap<String, String> to temporarily save the dictionary
5. Do all fault-tolerant processing that may occur in each place
6. For different situations that may occur, print different information in the ResultShow of the GUI
7. Use @ to split commands, words, and meanings, and use & to split multiple meanings, and display them line by line.

8. Conclusion:

The basic functions of the whole project can be realized. You can input words and meanings, save them in the dictionary (disk, memory), and then you can use the methods of adding new, searching, removing, and updating.

Each client corresponds to a connection and opens its GUI. The whole project also considers multi-threading and concurrency issues (using sockets and threads, also the ConcurrentHashMap and LinkedBlockingQueue)

However, there is no extension of additional functions, nor does it beautify the GUI page. For message exchange, self-design is used instead of JSON or serialization.

The overall project completion is very good. In the case of a single server and multiple clients, the dictionary effect with GUI is realized.