**Name: Siqi Zhou**
**Student ID: 903274**

**COMP90015: Distributed Systems**

**Semester 1, 2022**

**Assignment 1 Report**

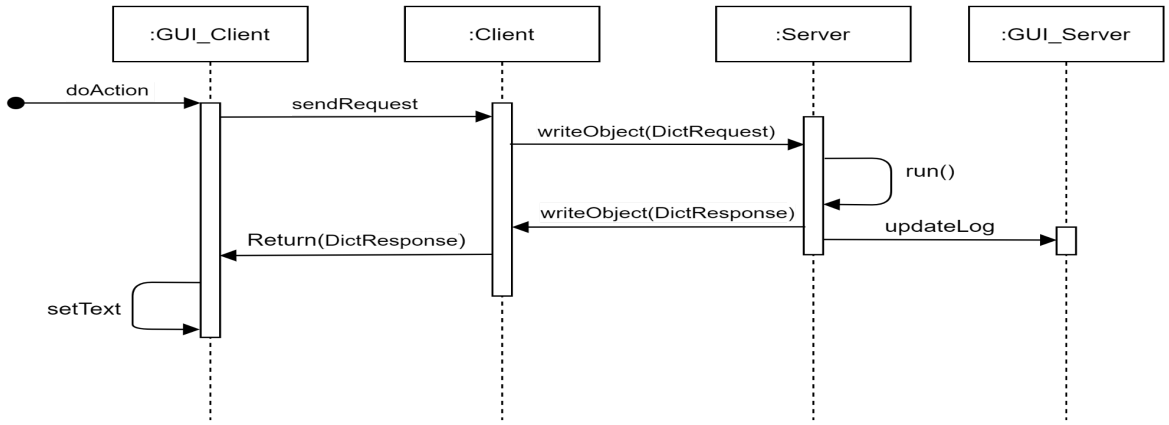
This report is based on a project which requires building a multithread dictionary with a client-server architecture. It will briefly overview the problem of the project, discuss the frontend and backend design and structures, as well as critically analyse the final result.

## 1. Problem Context

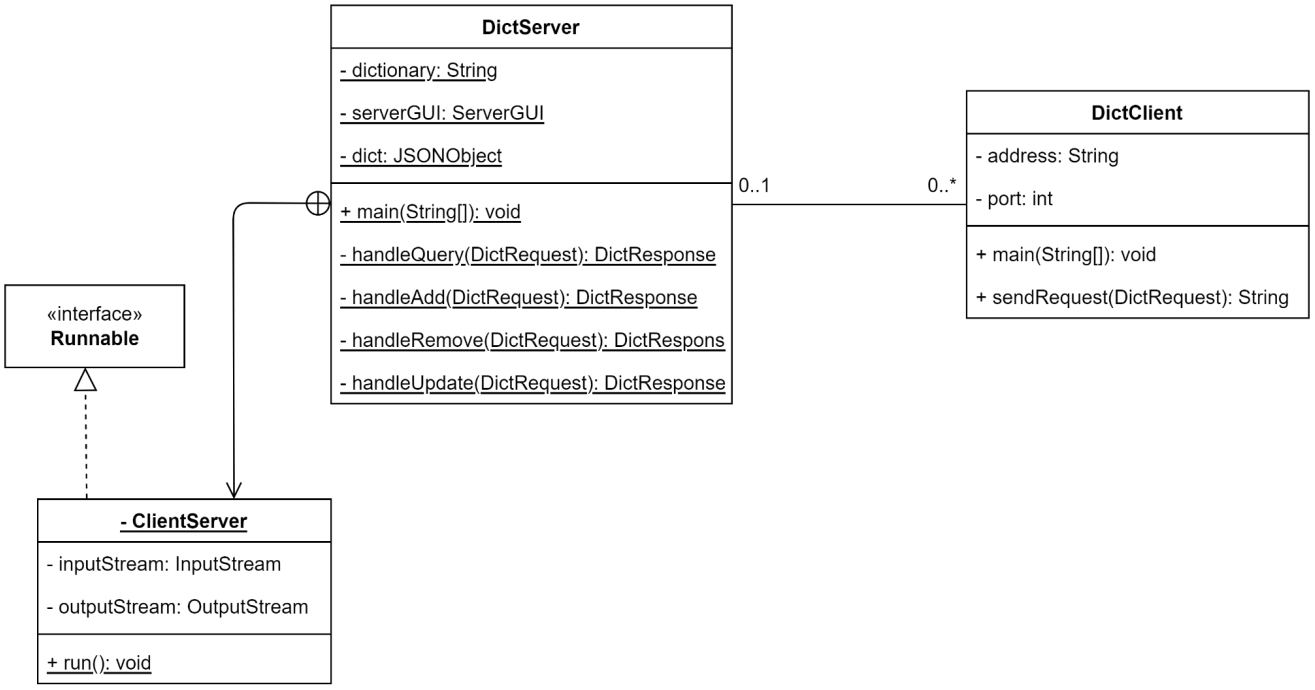
This project aims to build a multithreaded dictionary with client and server capabilities. There will be multiple clients operating simultaneously, and the server should be able to solve any concurrency issue. Moreover, this system should provide a user interface that allows users to perform all necessary actions. Therefore, there would be three main problems to consider. Firstly, it needs to choose a method of communication between the client and the server. This method includes appropriately choosing the socket protocol and the structure to transfer the information. Secondly, the management of threads also needs to be carefully considered as the system will be required to serve multiple requests simultaneously. Finally, the interface needs to be straightforward and provides some relative information to show the current status of the system or the results of each query.

## 2. Structure

As Graph 1 shows, The system follows the client-server architecture with two classes, one for the server side to handle any requests from clients and another for the client side to get input from the user and send it to the server through the designed java object. As graph 2 shows, one server can exist with any number of clients, and one client can link to no more than one server. A client can exist without a server because of the one thread per request architecture, but thus it will not be able to send any requests, and a corresponding error message will pop up when the client tries to send a request without a connected server. The server and the client use java object serialisation as the communication protocol. This is because the contents and types of data to exchange are fixed. Thus, the fields of these data can be pre-determined. Whatsmore, using java objects avoids dirty data since the server will throw the 'ClassNotFoundException' when receiving unexpected data. Besides, the named fields in java objects improve the readability of the code; it provides an intuitive understanding of communication contents.




Graph 1

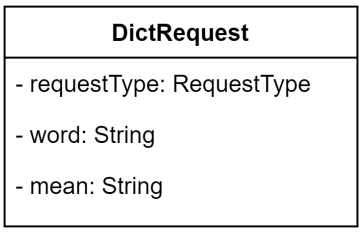



Graph 2

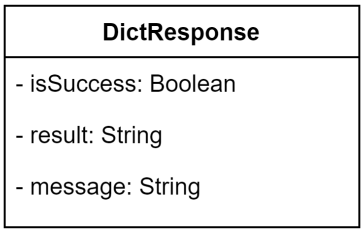
### 2.1 Dictionary

The dictionary uses the JSON (key, value) format to store items, with each word as a key and the meaning as the value. When the server is loaded, the data will be extracted from the dictionary file, converted into JSONObject, and stored in the memory. Moreover, every time an action that changes the content of the dictionary has been executed successfully, it only updates in the memory. However, every time the server is closed, all the changes will be updated in the dictionary file.

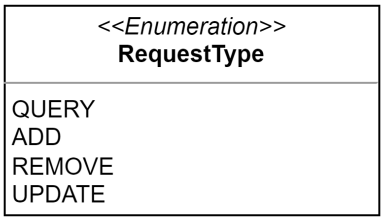
### 2.2 Request and Response




Graph 3.1




Graph 3.2




Graph 3.3

The two java objects, ‘DictRequest’ and ‘DictResponse’, are used to set the information to be exchanged among the server and client. As graph 3.1 shows, the ‘DictRequest’ object contains the current request type, which is an enum type as graph 3.3 displays, including ‘query’, ‘add’, ‘remove’ and ‘update’. This is used for the server to determine which method should be called to handle the coming request. The ‘word’ field is the word to conduct activities with, and the ‘mean’ field is the meaning of the word put in by the client, which is only filled when requesting the add or the update action. Graph 3.2 shows the structure of the ‘DictResponse’ object. The ‘isSuccess’ field represents whether the action is succeeded. The ‘result’ field only applies to the ‘query’ action to put the result of the query. The message field contains any

messages for the client side, such as “Success!” when the action succeeded or any related error messages when the action failed.

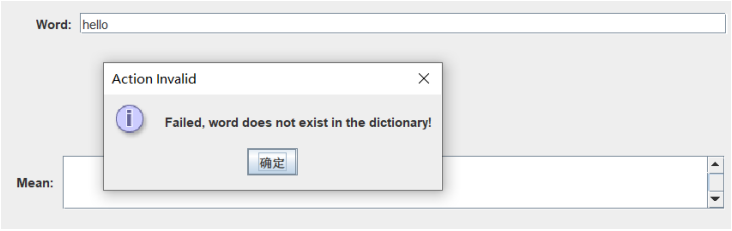
### 2.3 Server

The ‘DictionaryServer’ class provides the dictionary service; it handles all the requests from the client. When the server has been launched, the ‘main’ method will try to get the arguments and separate them into the port and dictionary path; it throws the ‘IllegalArgumentException’ if incorrect arguments have been provided. Next, the server tries to load the dictionary from the file path obtained from the argument into memory as a JSONObject. It catches the ‘IOException’ if read from file failed or the ‘ParseException’ if parsing dictionary contents into JSONObject failed. Then the server GUI is initialised. After getting all the required data, the server starts the socket and waits for any requests from the client. Once the server accepts a request from a client, as Graph 1 shows, it tries to get a thread from the thread pool and runs. Firstly, it waits for 0.1 seconds, the purpose is to avoid conjunction of the thread pool in case it is full. Then it creates a new ‘ClientServer’. The ‘ClientServer’ class implements the ‘Runnable’ interface. When this class has been executed, it gets the ‘requestType’ from the request, calls different methods to handle the request, and writes the response back to the client. It also gives an error message whenever there is an unrecognisable ‘requestType’. Finally, it provides the log information of current action to the server side user interface.

The four handler methods in the ‘DictionaryServer’ class each handle one kind of request. Once a handler method gets a request, it checks whether the request contains all the necessary fields. Secondly, it does the action requested towards the dictionary in the memory. Finally, it formats and returns a ‘DictResponse’ containing all the information to the client.

### 2.4 Client

The ‘DictionaryClient’ class deals with the actions of users, which is more general than the server class. This is because the more specific tasks correspond to user actions, which would be more appropriate to be processed by the buttons of each action type. When a client service is launched, it tries to read the server address and server port from the arguments. Then it initialises the client GUI. As Graph 1 shows, it contains a ‘sendRequest’ method; this will be used in the ‘clientGUI’ class to send a request to the server side. When called, it establishes a socket, sends the request to the server, and receives the response. When the response indicates that the request failed or catches any error, as Graph 3 shows, it appears a pop-up window with a related message in the interface to warn the user.

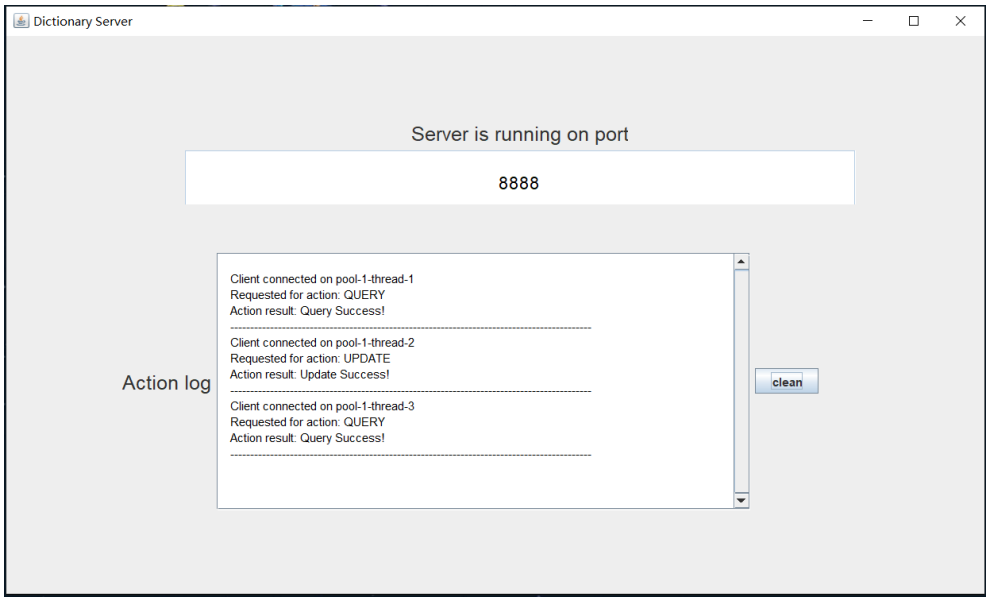



Graph 3

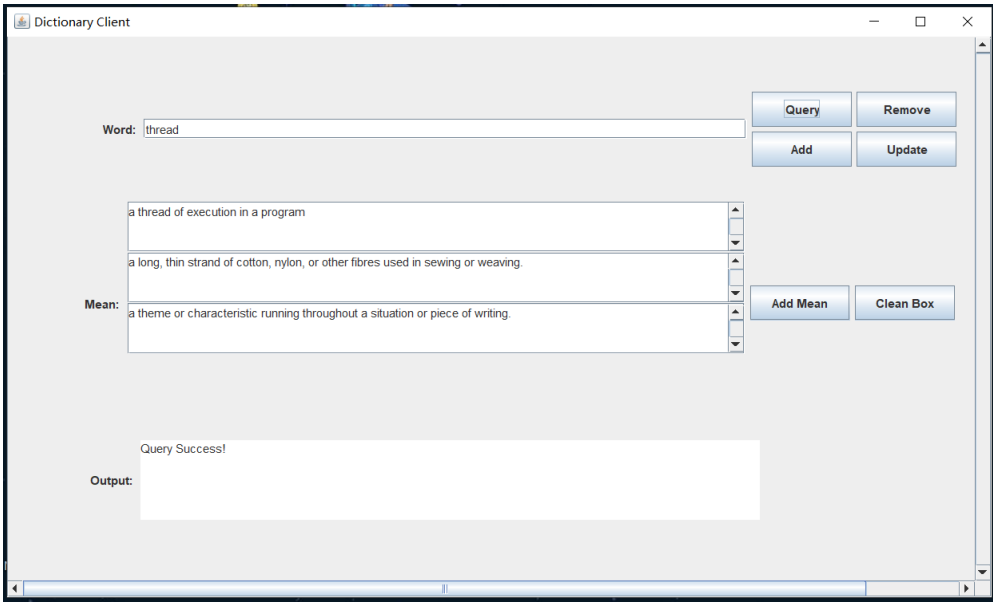
## 3. User Interface

The system contains two GUIs corresponding to each of the server and the client side. As Graph 3.1 shows, the server GUI shows the port that the server is running on. It also includes some related information, such as the request type and the processing result of each request in a log format, and the ‘clean’ button beside the log field cleans up the printed log information. Graph 3.2 shows the client GUI; it includes buttons for each action and text fields for user input, word meaning, and output information. When a user clicks on a button, it calls a corresponding action listener. The action listener first collects the text from the required input fields. For query and delete actions, it only gets a text from the ‘word’ field

and for the case of add and update actions, it also gets the ‘mean’ field text. Secondly, it forms a ‘DictRequest’ and calls the ‘sendRequest’ method in the ‘dictionaryClient’ class to get the response. Finally, it gets information from the response, including the meaning of words for the query action and the action output for all kinds of requests. The client interface also allows multiple meanings for one word. To add a word with more than one meaning, just click the ‘Add Mean’ button, an empty text field will show, and user can type in another meaning of the word. The ‘Clean Box’ button is used to clean up extra empty text fields. When doing query action, each meaning will also shown within a corresponding text field.




Graph 4.1




Graph 4.2

## 4. Socket and Threads

TCP is a connection-oriented protocol, and it only communicates when a connection has been established by the three-way handshake. On the contrary, UDP is connectionless and throws data without confirmation from the receiver. UDP provides better speed than TCP, but this project requires reliability instead of velocity. Therefore, the TCP protocol has been chosen to fulfil this requirement. The multiple

clients and the concurrency issue are also essential to be considered. Initially, the server uses the method of creating a new Thread instance each time receives a request, with the architecture of one thread per connection. However, infinitely creating new threads without any restrictions may cause a large amount of execution cost. Therefore, the 'FixedThreadPool' has been chosen to improve performance. The pool size has been set to the number of processors available as this will be the maximum capability of the server. Besides, the task queue provided by the 'FixedThreadPool' provides the ability to hold extra tasks when reaching the limit. However, to maximise the usage of threads in the thread pool, the one thread per connection architecture may result in more extended occupancy of each thread thus it changes to use the one thread per request architecture, as each thread will be released and able to be reused as soon as a request been finished, and the time requires for a request is much smaller than the time occupied by each connection.

## 5. Conclusion and Analysis

In conclusion, the current production of this project satisfies the basic requirements. It has a server-client structure and uses TCP sockets for connection. It uses threads with a thread pool to execute the actions. It considered the concurrency issue by setting a wait time before each execution, with the self-contained task queue from the 'FixedThreadPool' for extra tasks. It handles the error messages by throwing them when just started and uses pop-up messages when there is a user interface.
However, there are till some problems of the System. Firstly, the threadpool maximises the use of processors, which means it not truely saves memory. This is because the intended number of users are not given and it can be more deterministic if such estimations were given.Moreover, the one thread per connection architecture is missing some key information. For example, the client would not know if there is an existing server and vice versa unless they send an request to try to connect.

## 6. Creativity

- A server interface shows the current port of the server and scrollable requests logs with a cleanup button.
- A scrollable client user interface, with the ability to add any number of meaning fields for words with multiple meanings when doing 'add' action, or show number of text fields corresponding to number of meanings when doing 'query' action, with a cleanup button to remove empty meaning fields.