**Name: Ruofan**
**Surname: Zhang**
**Student ID: 1029050**

# COMP90015 2020 SM1 Assignment 1 Report

## Brief Description

In light of the assignment problem description, this project implements a client-server architecture with a thread pool to handle concurrent requests of dictionary services. Communication between client and server is completed by TCP sockets. Message exchange protocol is simple. When users enter inputs in GUIs, the name of the functionality such as “query”, “add” or “remove” and the word (and meaning, if any) will form a string formatted like <functionality>,<word>(,<meaning>). The string then is sent to server in socket. When server gets the string, it splits it by “,” and gets the functionality requested, word (and meaning). By the functionality requested, the server invokes corresponding dictionary services and sends back the result (see Dictionary Services of the Components of the System section).

## Components of the System

The system mainly consists of 4 parts, which are Swing Windows, Dictionary Services, Client and Server.

## Swing Windows

There are 11 swing windows in total. TemplateFrame is the parent of all the other 10 and it implements the basic settings such as window size, font, location, etc..

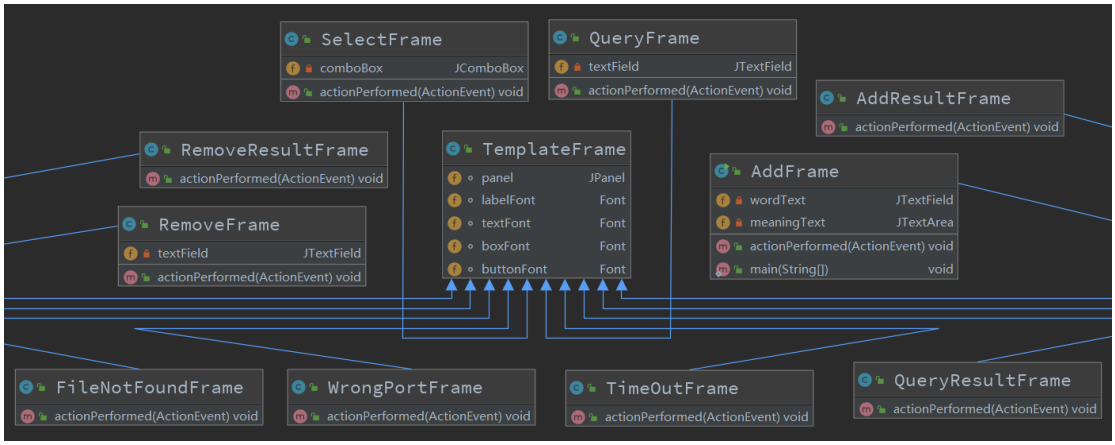


Figure 1

SelectFrame is the first window that users see after client correctly launched and it’s like this:

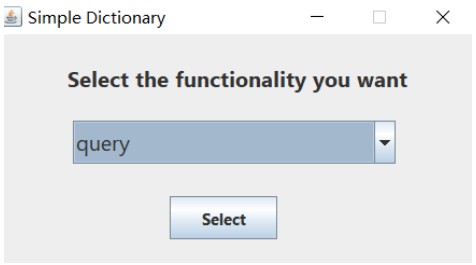


Figure 2

With a list of functionality to choose from:

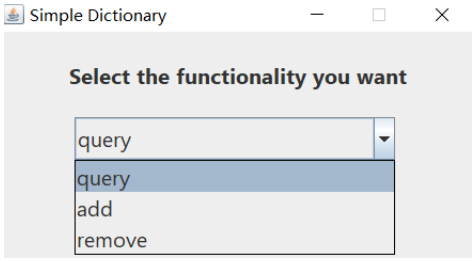


Figure 3

When users select “query”, it will go to QueryFrame, which is like:

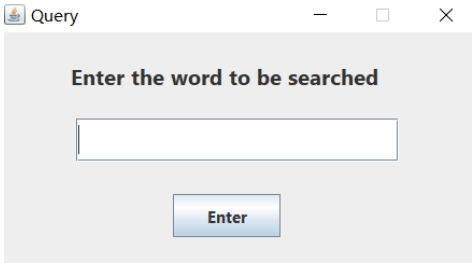


Figure 4

And if users enter the right input, it goes to QueryResultFrame, which is like:

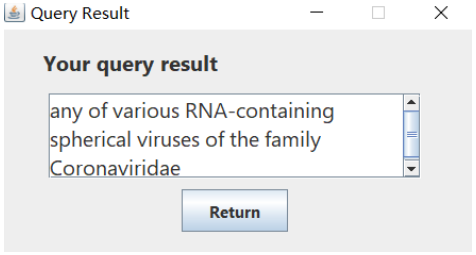


Figure 5

The following is AddFrame:

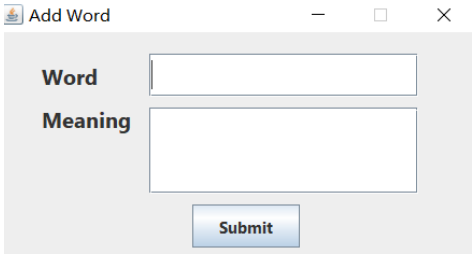


Figure 6

AddResultFrame is like:

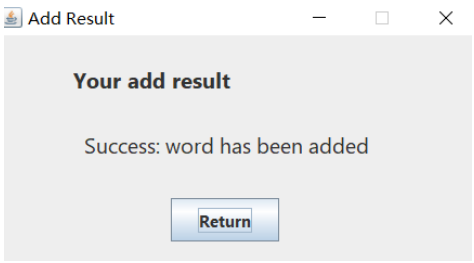


Figure 7

RemoveFrame is similar as QueryFrame and RemoveResultFrame as AddResultFrame.

When users click the button, QueryFrame, AddFrame and RemoveFrame will create a Client object carrying the inputs to communicate with the server and get responses. And then, they will create their corresponding result windows with responses displayed on them.

## Dictionary Services

In this program, dictionary is maintained in a normal text file and each entry in the dictionary follows the format: <word>,<meaning> and occupies a line as Figure 8 shows.

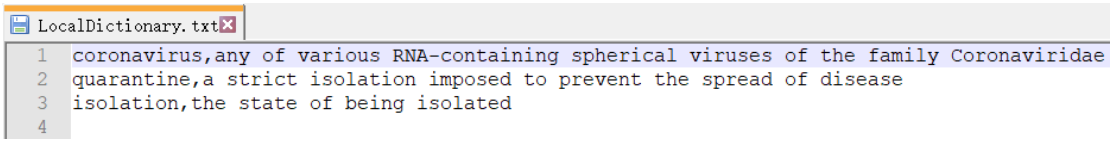


Figure 8

DictionaryService is an interface that defines the required functionality, which are implemented in DictionaryServiceImpl. ServiceResultEnum (Figure 10) is an enum class that defines responses to user requests. DictionaryServiceImpl takes arguments passed from Server and returns responses.

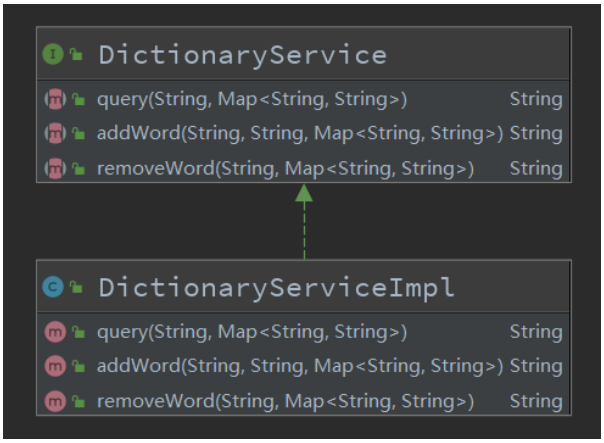


Figure 9

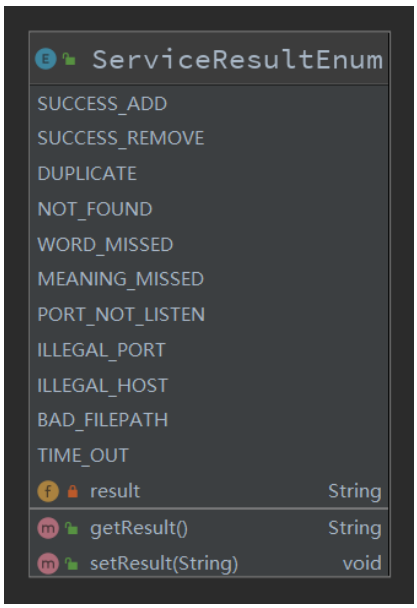


Figure 10

## Client

Client is responsible for using socket to communicate with the server and passing messages entered in windows. The method connectServer takes two String arguments. The first one is the name of requested functionality, for example, for query requests, the first argument would be “query”. And the second argument can be a word to be queried or removed, or can be string formatted as “<word>,<meaning>” when “add” function is requested, which is designed to keep in step with the dictionary file format (Figure 8).

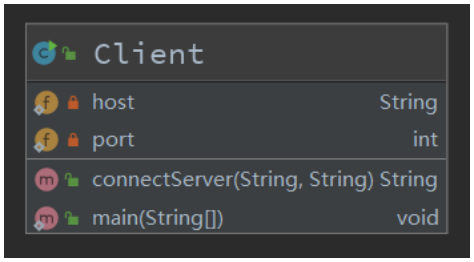


Figure 11

## Server

Server is responsible for listening to client requests and calling DictionaryServiceImpl with arguments from Client to handle the requests. After DictionaryServiceImpl returns the response, server will write the dictionary in memory into disk, if the dictionary has been modified, and send back the response. New windows would be generated to display the response.

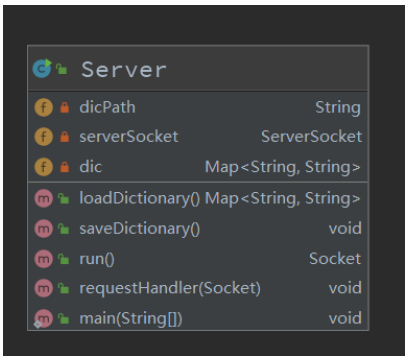


Figure 12

# Interaction Diagram

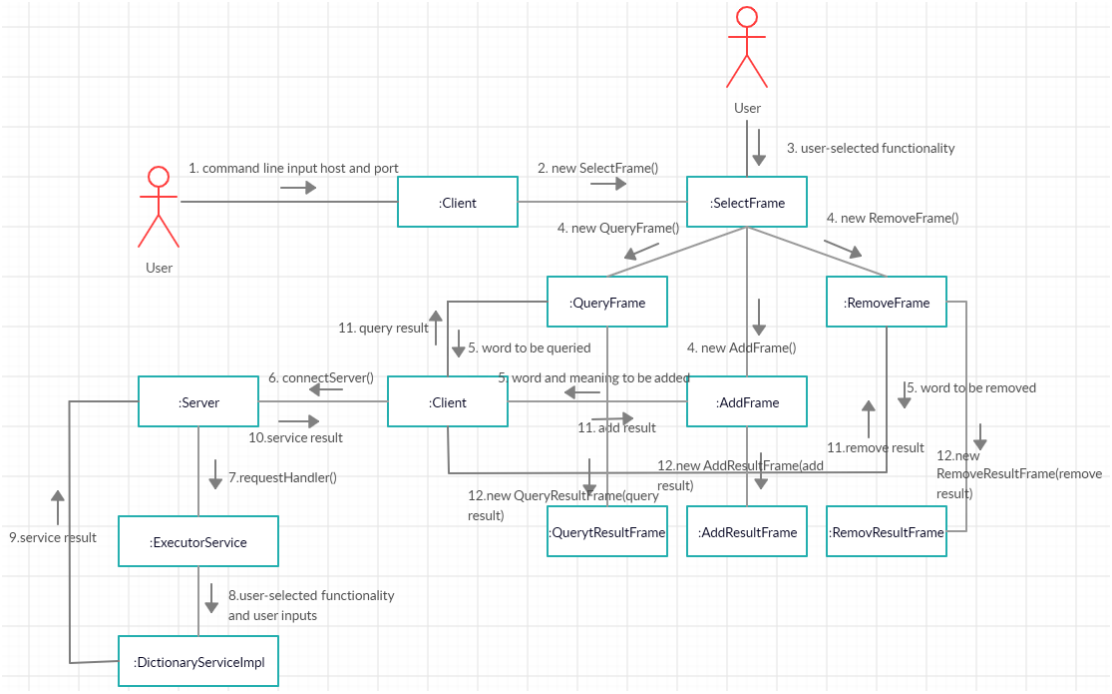


Figure 13

This diagram illustrates how this program completes the task given in the assignment problem description. At the start, the Server must bind a port and find the dictionary file on disk. If errors occur, corresponding prompts will arise as described in the Problem Context section. And then it loads the local dictionary and stores the word-meaning pairs in a hashmap in memory. It also implements a thread pool based on LinkedBlockingQueue to handle concurrent requests (see Analysis & Conclusion section for more details on the thread pool.) In the main method, Server will be run and keep listening to requests on the port.

On Client side, when Client is to be launched, the user has to enter designated host and port and prompts pop up if there is any problem. After both Server and Client were successfully launched, Client would generate a SelectFrame window object for the user to choose the wanted functionality. If the user chooses “query”, a QueryFrame object will show and AddFrame for “add”, etc.. And then in the QueryFrame window or RemoveFrame window, the

user will enter the word to be added or removed. In the AddFrame window, the user will have to enter both the word and meaning to be added. After that, when user clicks the enter button, a Client object will execute the connectServer method and use socket to send the inputs to server.

When Server has a new incoming request, the request will be handled by requestHandler method, which will be executed by a thread in the pool. The method handles the request by reading the inputs from socket and calls DictionaryServiceImpl to complete the service and return the response.

The response will be written into Server's socket with the Client and passed to entrance window object, i.e., if the user submits inputs in QueryFrame window, the QueryFrame object will have the response and then it will call its result window, a QueryResultFrame object, and pass the response to the object to be displayed on it. Now, a successful dictionary service request has been completed.

# Problem Context

The task is to implement a client and a dictionary server and ensure that requests launched by the client would be properly handled by the server. Therefore, problems might arise during the step of establishing connection and these may include wrong host or port number entered on both sides or the server shuts down while clients are running. To inform users when such problems occur, error information windows will pop up in this program. For example, if a user inputs an illegal port number, such as string instead of numbers, the program will prompt:

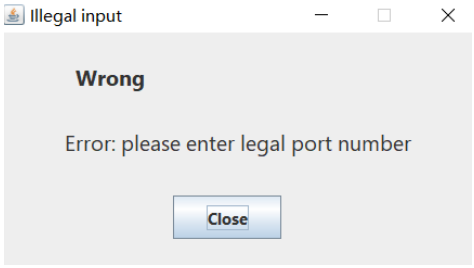


Figure 14

And when the connection times out, server shuts down or the client connects to a wrong host or wrong port, it prompts like:

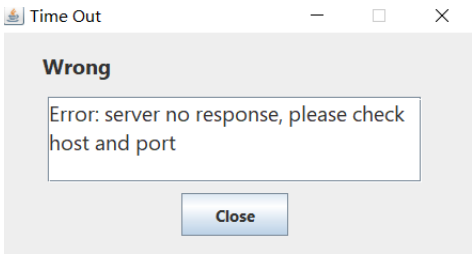


Figure 15

On the other hand, after both client and server launched successfully, a functionality selection window will pop up and users have to choose functionality that they need, i.e., query, add or

remove. And then users must enter the word to be queried or removed or enter word and meaning to be added. In this step, problems occur when users don't enter required fields. For example, if users don't enter the word, the program says:

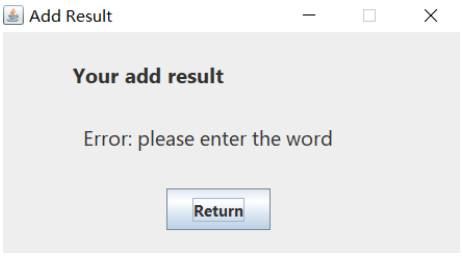


Figure 16

Or if meaning hasn't been entered, it shows:

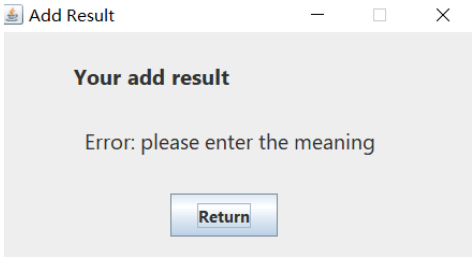


Figure 17

And the required prompts like "word doesn't exist" or "word has already existed" have been implemented.

Problems may also occur when server cannot find the local dictionary file. In this case, the following will show:

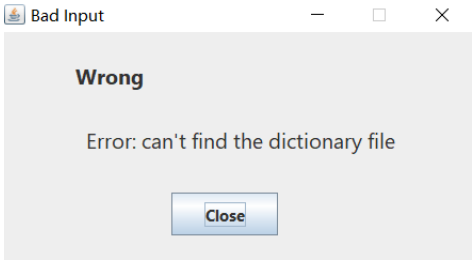


Figure 18

# Analysis & Conclusion

Advantages of this program may include the following elements:

a) Synchronization has been implemented in thread design to ensure the consistency, i.e., requests are handled based on the first come first served principle.
b) Inputs check is implemented before any connection is established. When users don't enter any required input or enter bad inputs, error information windows immediately show up without any input passed to server, which is designed to cut unnecessary overhead.
c) During the launch of the server, it reads local dictionary file on disk once and only once and stores the word-meaning pairs in a hashmap in memory. And then the query, add,

remove operations are done by manipulating the hashmap. Due to the property of hashmap, it guarantees good time complexity of these operations. When any word is added or removed, the server will write the dictionary into disk to ensure that the local file reflects the data structure in memory at any point in time.

d) Each request is executed by a thread in the thread pool, which is efficient in terms of managing threads. In this project, there are 5 core threads in the pool and the maximum pool size is 200 to handle a multitude of concurrent requests. If the current pool size is more than the number of core threads, extra threads would be destroyed right after being idle to save system resources. If concurrent requests exceed the number of core threads, the exceeding part would be queued in a LinkedBlockingQueue with capacity of 1024 to prevent out of memory incidents.

On the other hand, things that can be improved may include these:

a) In regard to user-friendliness, KeyListener should be implemented so that users can also use the keyboard to control the GUIs. And multi-language support is another good option.
b) This program cannot guarantee that bad inputs would not be added into the dictionary in cases where users submit non-character inputs or meaningless words that don't exist in the real world.
c) Thread pool size is set tentatively and is worth reasoning.
d) Since there is only a slight difference among QueryResultFrame, AddResultFrame RemoveResultFrame, TimeOutFrame, WrongPortFrame and FileNotFoundFrame, a general window template can be abstracted from these classes.

## Excellence Elements

Notification of errors is implemented when problems occur during the launch of both client and server, the establishment of connection and the user interaction with the GUIs (see Problem Context section).

## Creativity Elements

This program allows users to enter words or meanings with uppercase, lowercase or a combination of them when requesting query, add or remove services, which is achieved by the fact that when server gets the inputs from client, it converts all inputs to lowercase and all operations are executed on this basis. For example, user A wants to add the word "STUDENT" and the meaning "People who go to school every day." The server will parse them into lowercase and store them into the dictionary. If user B wants to query "Student," the serve also parses it as "student" and looks for the meaning in the dictionary. Finally, user B will get the response as "people who go to school every day."