COMP90015 Distributed System
Assignment 1 Report

THE UNIVERSITY OF

MELBOURNE

Quang Trung Le (987445)

22 April 2020

1 Introduction

1.1 Aim

The main objective of the project is to overcome the problem of multi-threaded dictionary server. In more details, a multi-threaded server allows many clients to connect concurrently. Each client can search the meaning, add new words and remove an existing word. Besides, a user interface (GUI) is also considered in this this project. In other words, this project focuses on designing:

- Architecture: client - multithreaded server
- Tasks: searching - adding - removing
- User interface on clients

1.2 Dataset

A dictionary data file named "dictionary.txt" will be the dataset in this project. Assume that the server is this project is an English dictionary server, the file will contains roughly 600,000 words, which is the English vocabulary size [1]. However, because the file was not provided, a new data file is created.

2 System Analysis

2.1 Architecture

According to the requirement, this project uses a multi-threaded server. Then, the architecture includes thread-per-request, thread-per-connection, worker pool. In worker pool, a fixed pool of threads are created, which means each thread is in charge of a task (searching, adding, and removing). Meanwhile, thread-per-connection creates a new thread for each client connection, which is convenient when the number of clients increase. Therefore, this project uses thread-per-connection architecture.

2.2 Interaction

Firstly, this project requires reliable communications between clients and server. Therefore, TCP is a reasonable choice for socket.
Secondly, in any sending message, standard protocols and data format are essential. In this aspect, JSON - a popular data interchange format - is a solution. However, the protocol for messages client-server and server-client are different. Fundamentally, the information that needs to be exchanged between clients and server includes:

- *{"command": <command-data>}*: the task that clients want to conduct.
- *{"vocabulary": <vocabulary-data>}*: the word that clients want to search/add/remove.
- *{"meaning": <meaning-data>}*: the meaning of the word that client want to add or the meaning returned by "search" task.
- *{"announcement": <announcement-data>}*: the announcement on the task status (success/ fail).

Nonetheless, not all the data are necessary. For example, in the message from clients to server, command, word and meaning are necessary if clients want to add a word, but the meaning is abundant in searching or removing tasks. Similarly, in the message from server to clients, <announcement-data> is essential in all the tasks, but only searching task needs the <meaning-data>. However, for simplification, typical protocols are used for all the tasks.

- Client to sever:
  {"command":<command-data>, "vocabulary":<vocabulary-data>, "meaning":<meaning-data>}
- Server to client:
  {"meaning":<meaning-data>, "announcement":<announcement-data>}

3 Implementation

3.1 Class Diagram

3.1.1 Server

According to the requirements of the project, there are three functional tasks: searching, adding, removing, which require two additional IO tasks: dictionary reading and writing. Because they are specific services of dictionary server, they are executed inside "PlainDictionaryService" class. Meanwhile, two classes "DictionaryServer" and "MultiThreadsDictionaryServer" are responsible for all the tasks that relate to communicating.
The functionality of all the classes related to server:

- "PlainDictionaryService": conducts three functional tasks.
- "DictionaryServer": receives the incoming message, assigns the functional task, sends the returned message.
- "MultiThreadsDictionaryServer": initialize the connection (socket), create threads.

The UML diagram of server's components is shown in figure 1.

3.1.2 Client

Clients conduct two main tasks: sending requirements and receiving the returned results, which are executed in "DictionaryClient". Additionally, a user interface is created in order to simplify the input procedure for clients. As a result, all the data for the sending message come from the user interface Actions in "DictionaryClientGUI". The UML diagram of clients' components is shown in figure 2.

3.2 Failure and Exception Handling

In the whole process, there are some main stages: client connects to server; client sends the data to server; server processes the incoming data; server sends the returned data to client; client shows the returned output. During these stages, many errors and exceptions need to be handled by either server or client. Some possible failure are shown in table 1 (# means "the number of").

1. Console input issues:
They are major issues, which causes the missing of the most important information to initialize

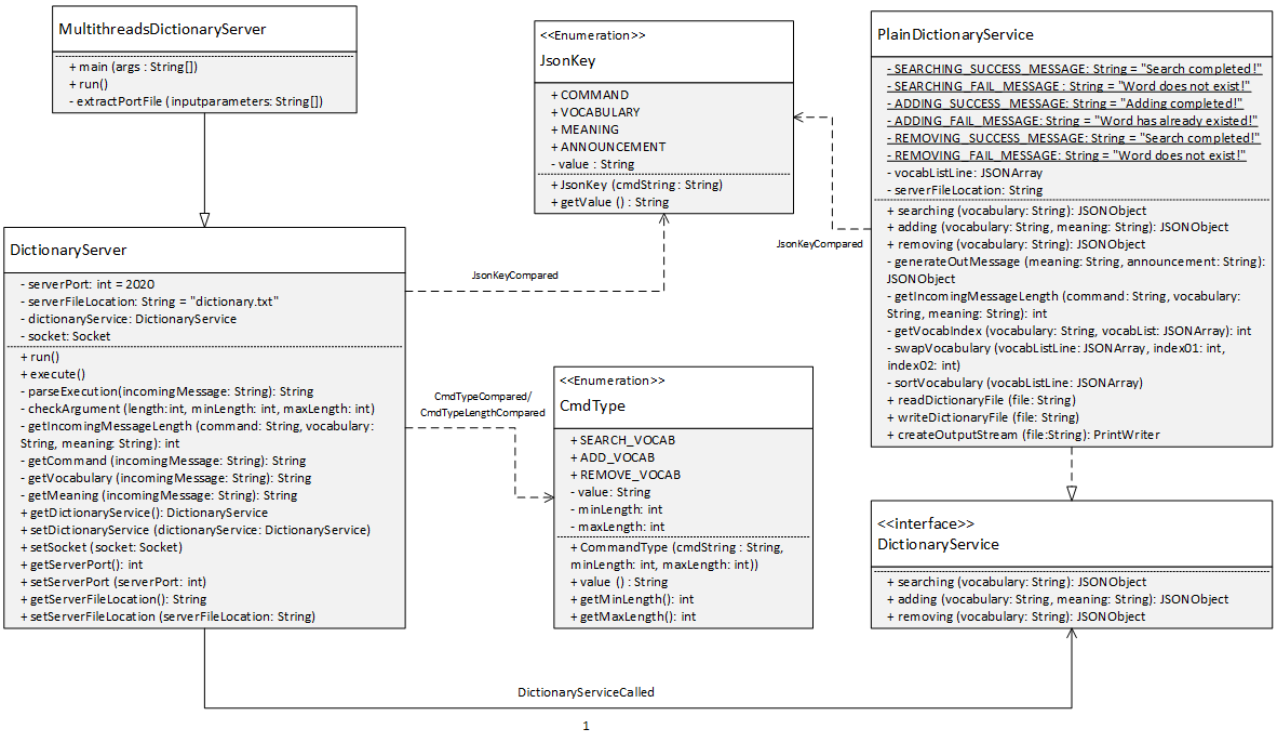


Figure 1: UML diagram of Dictionary Server

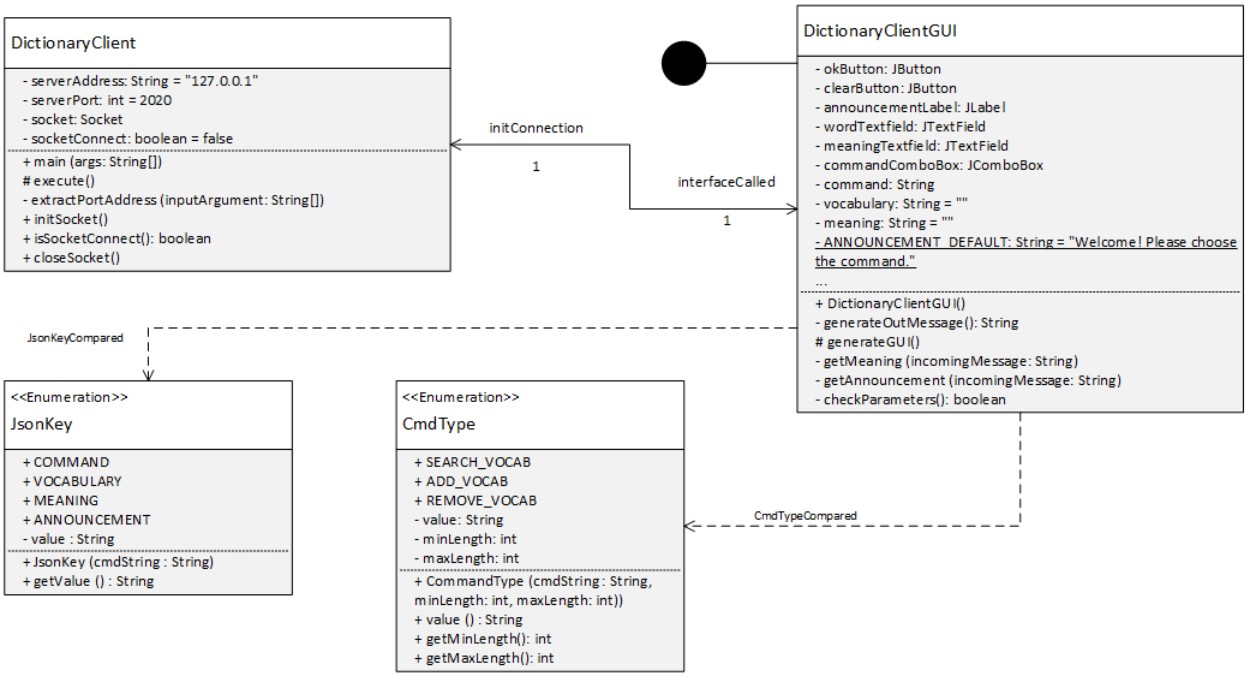


Figure 2: UML diagram of Dictionary Client

Failure	Client	Server
Console inputs	Illegal # input parameters Illegal input data type	Illegal # input parameters Illegal input data type
Communication	Address not reachable	
Functional Tasks		Searching word not found Adding existing word Adding word without meaning Removing word not found
IO exception		Dictionary does not exists Dictionary cannot be created Dictionary cannot be written

Table 1: Possible failures during execution

server-client connection. When these issues occur, an *IllegalArgumentException* is thrown and system exits.

2. Communication issues:
If "Address not reachable" occurs, the service cannot run. Then, *IllegalArgumentException* is thrown and system exits.

3. Functional task issues:
Although most of the functional task failure occur in server, these issues come from the inputs from clients. Therefore, they will be announced to clients via the <announcement-data> and displayed on GUI. Some examples are shown in 3.3.

4. IO exception:
These issues happen during reading/writing process, as part of functional task. If "dictionary.txt" does not exist, an "FileNotFoundException" is thrown an new file is created. If the file cannot be created, the program stops.

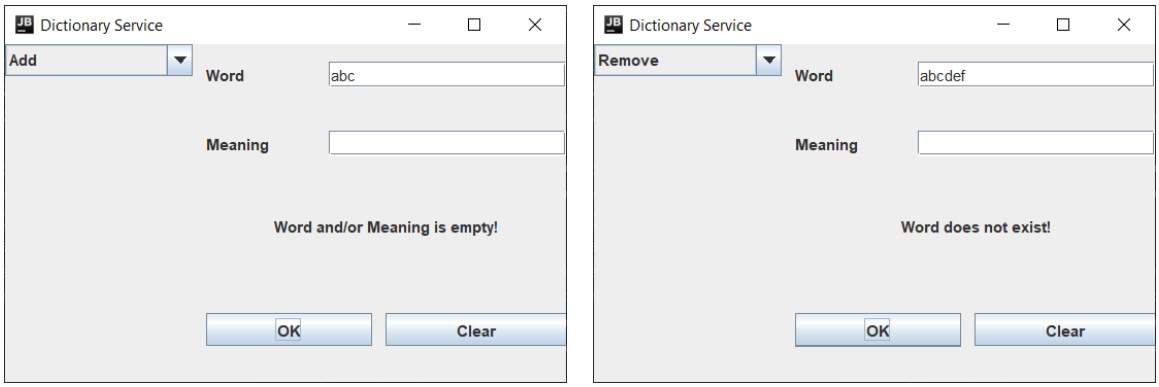
3.3 User Interface

As mentioned in 3.1, the purpose of this interface is to simplify the input procedure. In other words, user will select the command (task), input the word and meaning. Besides, some failures caused by the users, such as "adding word without meaning" or "Searching word not found", should be announced for users' awareness. Some examples can be seen in figures 3.

4 Conclusion

This project successfully deals with simple dictionary tasks such as: adding word, searching word and removing word, using multi-threaded server. Besides, a part from popular issues mentioned in 3.2, this project also covers "Socket not connected" or "Invalid command" (command is empty) although these issues hardly occur.

However, there are some weaknesses. Firstly, the dictionary file are loaded and written in every dictionary service (searching/adding/removing), which consumes a large amount of computer re-



(a) GUI with issue Word/Meaning missing (b) GUI with issue Word not exist

Figure 3: Some popular issues of dictionary service

source. However, because this project use the thread-per-connection architecture, and socket closes immediately after the message is sent to clients, this issue is not really a disadvantage.

References

[1] B. Aarts, S. Chalker, and E. Weiner, *The Oxford dictionary of English grammar*. OUP Oxford, 2014.