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

Kaixun Yang, 1040203

The University of Melbourne

April 17, 2021

1. Problem Context

In this project, a multi-threaded dictionary system server that allows concurrent clients to use needs to be designed and implemented using the client-server architecture. Clients can use four functions of this dictionary system, including querying the meaning(s) of a given word, adding a new word, removing an existing word and updating meaning(s) of an existing word. Errors on both the server and the client side should be handled properly, such as I/O to and from disk, network communication and input arguments from the console. In addition, a Graphical User Interface (GUI) is required for this project.

2. System Components

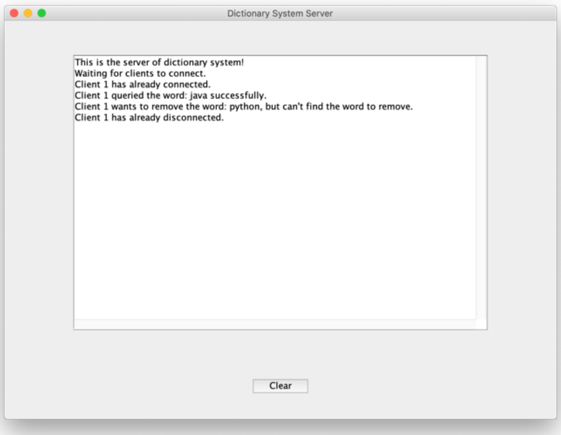
The dictionary system consists of four parts: the server, the client, the GUI for server and the GUI for client.

2.1 Server

Server is implemented in the Class Server. It handles requests from clients and offer responses. It uses TCP as the protocol which can guarantee the reliability. The server handles the multi-threaded clients by thread-per-connection. In order to ensure the consistency, the server uses keyword synchronized in the declaration of the static method *useDictionary()*. The operation logs are displayed on the server GUI. When closing the GUI, the server will save the changes during runtime to the local file. The server will handle the exceptions and display them on the command line. For the non-compliant operations of clients, the server will reply "Error: detailed message" to the clients.

2.2 Server GUI

Server GUI is implemented in the Class ServerGUI. It's a static class, which offers the static method *setGui()* for Class Server to start the GUI for server. The server GUI will show the operation logs, such as which client has connected, which client has disconnected, which client modified the dictionary. The server GUI also offers the clear button to clear the logs on the GUI. It will also listen for window closing to call *saveDictionary()* in Class Server.



```
This is the server of dictionary system!
Waiting for clients to connect.
Client 1 has already connected.
Client 1 queried the word: java successfully.
Client 1 wants to remove the word: python, but can't find the word to remove.
Client 1 has already disconnected.
```

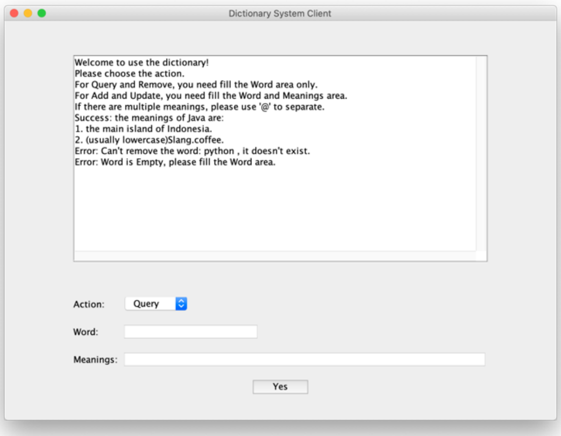
Figure 2-1: Server GUI

2.3 Client

Client is implemented in the Class Client. It sends requests from users to server, and handles the responses from server, then shows it on the client GUI. It will handle the exceptions and display them on the command line. For some illegal inputs of users when use the dictionary, it will show the error messages on the client GUI. Sometimes, client will not even send messages to the server if errors can be found on the clients' side.

2.4 Client GUI

Client GUI is implemented in the Class ClientGUI. It's a static class, which offers the static method *setGui()* for Class Client to start the GUI for client. The client GUI will show the responses from server and the error messages in clients' side. It can receive one choice (action) and two inputs (word, meanings) from the user, and offer a yes button for user to send messages to the server.



```
Welcome to use the dictionary!
Please choose the action.
For Query and Remove, you need fill the Word area only.
For Add and Update, you need fill the Word and Meanings area.
If there are multiple meanings, please use '@' to separate.
Success: the meanings of Java are:
1. the main island of Indonesia.
2. (usually lowercase)Slang.coffee.
Error: Can't remove the word: python , it doesn't exist.
Error: Word is Empty, please fill the Word area.
```

Action: Query

Word:

Meanings:

Figure 2-2: Client GUI

3. Implementation Details

3.1 Class Design

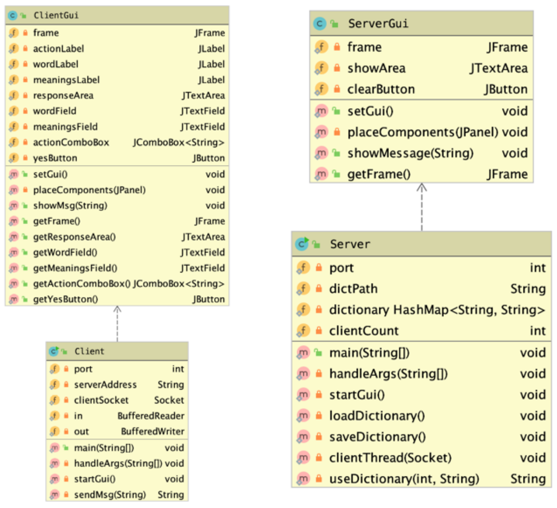


Figure 3-1: Class Diagram

There are four classes in the dictionary System.

Class Server is responsible for starting the server, opening the Server GUI, accepting clients' connections, processing the clients' requests and replying, maintaining the dictionary file, capturing and handling exceptions.

Class Client is responsible for connecting to the server, opening the client GUI, processing users' inputs from the GUI and sending requests to the server, receiving and processing the server's responses, showing responses to users through the GUI, capturing and handling exceptions.

Class ServerGUI is responsible for the layout of GUI and provides static methods to the server, including starting the GUI and displaying messages.

Class ClientGUI is responsible for the layout of the GUI and provides static methods to the client, including starting the GUI, obtaining users' inputs, and displaying messages.

3.2 Interaction Diagram

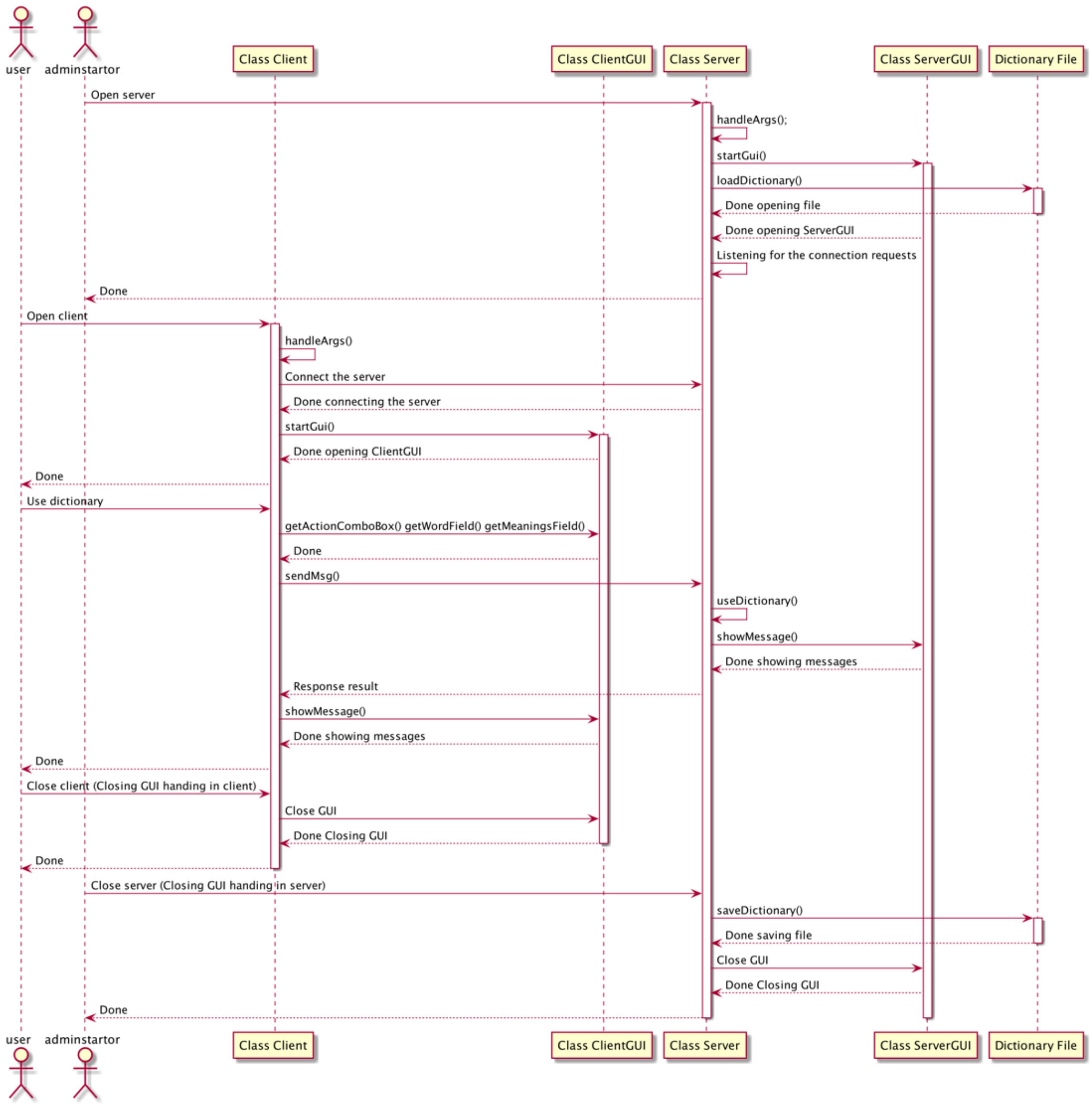


Figure 3-2: Interaction Diagram

3.3 Important Modular Design

3.3.1 Handle command line input

handleArgs() receives the parameters of the command line, which is used to check the number of parameters, the types of parameters, and whether the data range of the parameter meets expectations. If not, an exception is thrown. Both server and client have this function.

3.3.2 Load and save the dictionary file

The server stores the data by using Java Object Serialization into HashMap object. Key is the word and value is the meaning(s). *loadDictionary()* gets the HashMap object in file, if the file doesn't exist, it will new an empty HashMap. *saveDictionary()* stores the HashMap object in file when closing the server, if the file doesn't exist, it will create a new file first.

3.3.3 Handle multi-threaded clients

Use the while true loop to accept connection, and create a new thread for it and run it, using lambda expression *new Thread(() -> clientThread(clientSocket)).start()*.

For each thread, iteratively, *clientThread()* will handle the message from client and get the response by calling *useDictionary()* and send it to client, then calling *showMessage()* to show on the ServerGUI.

3.3.4 Handle requests from client

The request from client is a String, whose format is "action&word&meanings". Using *split()* to get each part in the String. Use action to determine whether to query, remove, add, or update, use word to query or remove, use word, meanings to add or update. After performing the operation, send response to the client. For the word, use *toLowerCase()* to ignore the difference between lowercase and uppercase of the word. then send the response to the client.

3.3.5 Handle concurrent operations

The server uses keyword *synchronized* in the declaration of the static method *useDictionary()*, which ensures the consistency of data.

3.3.6 Handle the actions of GUI

By using *addActionListener()* and *addWindowListener()* to listen for the actions of GUI button and GUI window, the server or client will perform the corresponding operations.

3.3.7 Handle the multi-meanings of a word

The format of meanings is "meaning1@meaning2@meaning3", so the client uses *split()* to get multi-meanings of the word and call *showMsg()* in the ClientGUI iteratively to show every meaning of the word in the GUI.

3.3.8 Handle input from the client GUI

Use getActionComboBox().getSelectedItem().toString() to get the action, use getWordField().getText().trim() to get the word and meanings without space in the head and tail. According to the format of "action&word&meanings", assemble them and send to the server by calling *sendMsg()*.

3.4 Failure Model

Side	Error Description	Error Message
Server	The length of arguments is not two.	Error: Wrong number of arguments, need two arguments Port number and Path of dictionary file(txt).
Server	The types of arguments are wrong.	Error: Wrong type of arguments, need Port number to be Integer and Path of dictionary file(txt) to be String.
Server	The value of port is not in the domain.	Error: Wrong value of Port number, need Port number to be between 1024 and 65535.
Server	The port has already been binded.	Error: Port number has been used, need to change another one.
Server	The IOExceptions occur in the communication between server and clients.	*System.out.println(e.getMessage())*
Server	The IOExceptions and ClassNotFoundException occur in the opening the file.	*System.out.println(e.getMessage())*
Server	The IOExceptions occur in the saving the file.	*System.out.println(e.getMessage())*
Server	Empty word.	Error: Word is empty, need a word to Query or Remove.
Server	Empty meanings for add and update.	Error: Word or/and meanings are empty, need a word or/and meanings to Add or Update.
Server	No word to query.	Error: Can't find the word: " + word + " , you can try to add the word into the dictionary system.
Server	No word to remove.	Error: Can't remove the word: " + word + " , it doesn't exist.
Server	Duplicate word to add.	Error: " + word + " has already exits, you can try to update the word.
Server	No word to update.	Error: Can't find the word: " + word + " to update, you can try to add the word.
Client	The length of arguments is not two.	Error: Wrong number of arguments, need two arguments Port IP address of server and Post number.
Client	The types of arguments are wrong.	Error: Wrong type of arguments, need IP address of server (String) and Port number (Integer).
Client	The value of port is not in the domain.	Error: Wrong value of Port number, need Port number (Integer) to be between 1024 and 65535.
Client	UnknownHostException occur in the connecting the server.	Error: Invalid hostname for server, please try other server Address or port.
Client	ConnectException occur in the connecting the server.	Error: Connection refused, please try it later or try other server Address or port.
Client	The IOExceptions occur in the communication between server and clients.	Error: Can't get I/O streams, please try it later.
Client	Empty word.	Error: Word is Empty, please fill the Word area.
Client	Empty meanings for add and update.	Error: For Query or Remove, please keep the Meanings area empty.
Client	Not empty meaings for remove and query.	Error: For Add or Update, please fill the Meanings area.

4. Critical Analysis

4.1 Advantages

- Saving all the changes in the dictionary file when closing the server is a good idea, because it avoids frequent file reading and writing.
- Using format Java Object Serialization to store dictionary file is a good idea, because it can be encrypted and compressed, which is secure and easy to use by java.

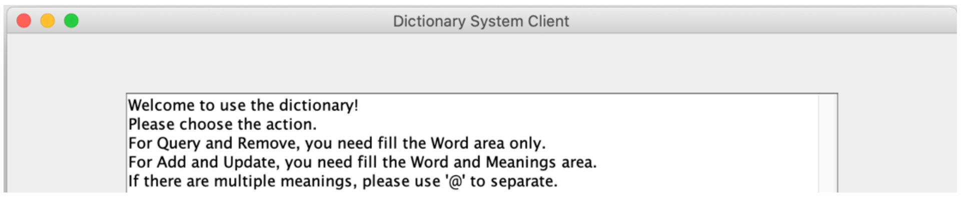
4.2 Disadvantages

- Thread-per-connection won't be a good choice for a large number of connections, because the repeated creation, destruction, and switching of threads cause a large overhead. It should use the thread pool for a large number of connections.
- Using TCP to be the protocol will consume more time. Because before transferring data, a connection must be established. I can use UDP protocol with a reliable communication infrastructure to improve the performance of consuming time.
- Using String to communicate between the server and client can lead some errors when split the String messages (such as there is a '&' in meanings). I can use JSON to avoid it.

5. Extra Elements

5.1 Excellence Elements

- Complete report with proper graphs (Class Diagram and Sequence Diagram) and analysis of my implementation's pros and cons.
- Good guidance for user in GUI, there are some initial introductory sentences in the text fields in Client GUI.



Welcome to use the dictionary!
Please choose the action.
For Query and Remove, you need fill the Word area only.
For Add and Update, you need fill the Word and Meanings area.
If there are multiple meanings, please use '@' to separate.

Figure 5-1 Guidance

- Good exceptions handling and proper notifications of errors to help user understand where went wrong, the details are in Section 3.4.

5.2 Creativity Elements

- An additional implementation of GUI for the server, which offers the information about which client connects and disconnects, what the actions and results of a clients' operations.
- I store the dictionary information in the hash map, which will take O (1) time complexity to query a word, make the requests more efficient. Besides, the multi-meanings of a word are showing in a point-by-point format which makes user have a better experience.