THE UNIVERSITY OF

MELBOURNE

COMP90015 Distributed System

Assignment 1

Multi-threaded Dictionary Server Report

Joseph Liu
1101805

6th April 2023

A. *Problem Context*

The problem context for this assignment is to design and implement a multi-threaded server using sockets for network communication. The server should allow concurrent clients to search for the meaning(s) of a word, add a new word, and remove an existing word. The objective is to demonstrate the use of sockets and threads as the lowest level of abstraction for network communication and concurrency.

B. *Socket, Threads and Components*

In this project, both the server and the client code use TCP sockets to make connection.
The server uses ServerSocket and Socket to listen and accept incoming client connections, and the client uses a Socket to connect to the server. Implementing TCP socket to make connection could guarantee the communication is reliable. This is because TCP is a connection-oriented protocol that provides error detection and correction mechanisms to ensure that data is transmitted and received correctly.

Furthermore, the server is designed to provide multithreading service to handle multiple requests from numerous clients at the same time without waiting for any single request to complete. To accomplish this, the code employs the thread-per-connection model, where a thread object is created using the MyRunnable() class, which implements the Runnable interface, whenever a client connects to the server. This architecture enables the creation of a new thread to manage the communication with each new client connection. As a result, the server can concurrently handle multiple clients, with each client assigned to a separate thread, allowing the system to operate efficiently.

C. *Class design*

This multithreading project consists of twelve classes designed to work together seamlessly. At the core of the project is the server class, MultithreadedServer, which manages connections with clients. The client class, Client, interacts with the server to access the dictionary application. The thread class, MyRunnable, runs in the background to handle multiple client connections simultaneously. The Dictionary class provides necessary methods for managing data of the dictionary within the project. In addition, there are eight GUI-related classes that help to create a user-friendly interface. Five of these classes extend JDialogue, while the remaining three extend JFrame. Together, these classes form a cohesive system that allows for efficient and intuitive multithreaded communication (refer to Appendix A Figure 5).

1. *Multithreading Server and MyRunnable*

The server class in this multithreading project performs several key tasks. It listens on a specified port, accepts incoming client connections, and creates a new thread to handle each connection. Before waiting and listening for client connections, the server first creates a Dictionary object and loads the existing file that stores dictionary data. This Dictionary object is then sent to the MyRunnable thread object, which is responsible for handling each thread connected by clients. By sharing the same Dictionary object among all clients, updates to the dictionary can be made concurrently, ensuring that every client receives the most up-to-date dictionary data.

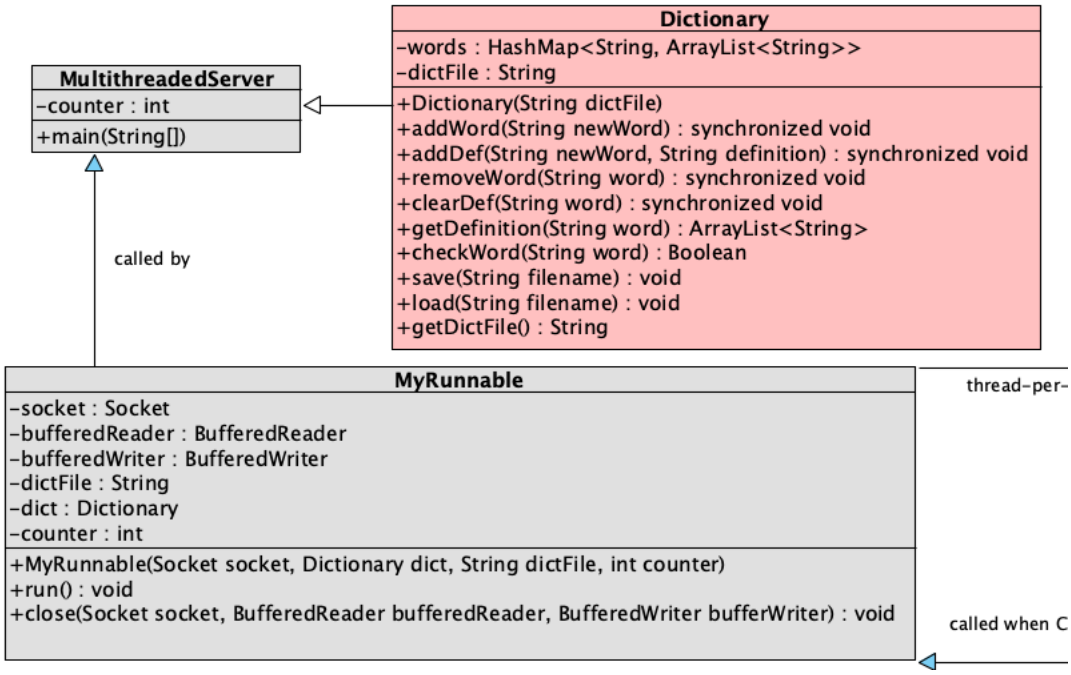

Figure 1. *UML Diagram - MultithreadedServer and MyRunnable*

2. *Client*

The Client class is an implementation of the client-side communication for a dictionary server in a multithreading project. It establishes a connection to the server and sends commands to it. Upon receiving user input, if the command is "d", the Client sends the command to the server, which then opens the MainFrame window to display the dictionary. If the command is "exit", the Client closes the connection and terminates gracefully. The Client class provides methods to handle input/output streams, manage the socket connection, and close resources properly. Overall, it serves as the interface between the user and the dictionary server, allowing for seamless communication.

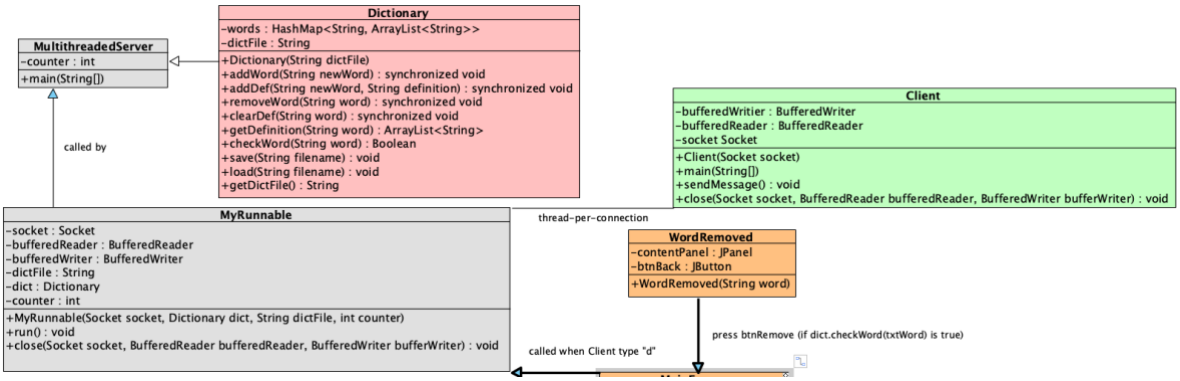

Figure 2. *UML Diagram - Client*

3. *Dictionary Class*

The Dictionary class is a fundamental component of the multithreading project and is responsible for storing and managing word definitions. It achieves this by using a HashMap instance to store each word along with its definition(s), where the data type for word is String and the data type for definition(s) is ArrayList<String> since there might be one or more definitions for a word. Additionally, the Dictionary class provides a set of methods that enable the addition and removal of words and their definitions, checking the existence of a word, retrieving the definition of a word, and saving and loading data.

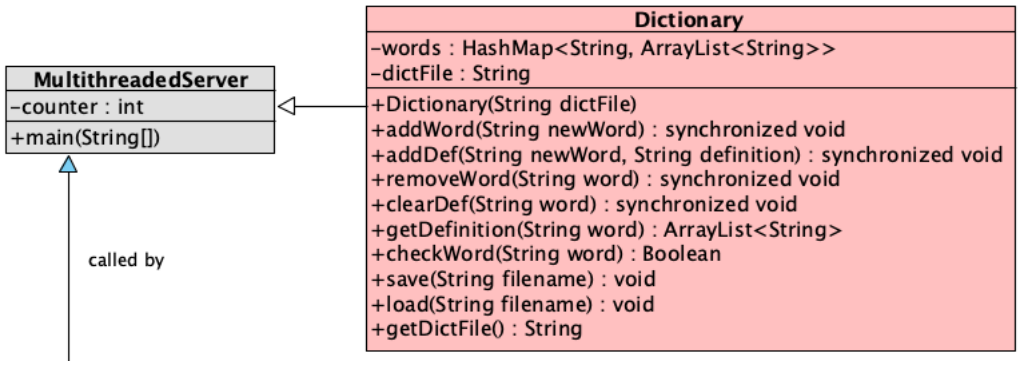

Figure 3. *UML Diagram - Dictionary*

4. GUI-related classes

There are eight GUI-related classes in this multithreading project. When a client types the command "d" to access the dictionary application, a MainFrame object is created, and the dictionary application window is displayed. In the main window, there is a JTextField called txtWord that allows users to type the target word and four JButtons, btnSearch, btnAdd, btnRemove, and btnUpdate, that allow clients to perform corresponding actions. All modifications made by users will change the Dictionary object dict, and all users can see any changes in real-time, which perfectly demonstrates the nature of multithreading. If a client presses the red exit button, the frame window will be disposed of, but the client will not be disconnected. Clients can choose to revisit the dictionary application by typing the "d" command in the command line, or they can type "exit" to disconnect from the server.

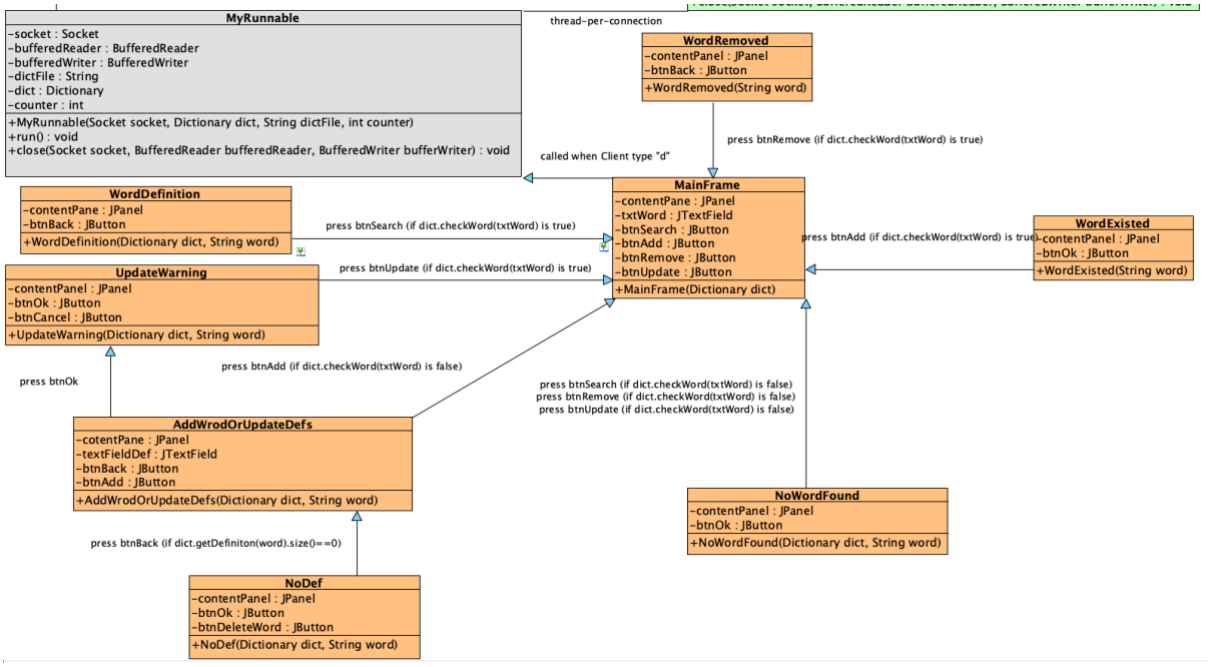

Figure 4*. UML Diagram - GUI (using WindowBuilder)*

5. Error Handling

The project implements several error handling techniques to ensure the smooth execution of the code. One such technique is the check for command line errors, which verifies if the number of command line arguments is correct and prints a usage message if not. In the case of the server, the usage message would be "Usage: java -jar DictionaryServer.jar <port> <dictionary-file>", and for the client, it would be "Usage: java -jar DictionaryClient.jar <hostname> <port>".

Moreover, an IllegalArgumentException is handled if the port number is out of range, i.e., less than 0 or greater than 65536. Additionally, an IOException and FileNotFoundException are caught, and if a file is not found, the program takes appropriate actions.

In case a user types an invalid port number, an error message is displayed, "Invalid port number: <port>". Furthermore, SocketException and UnknownHostException are caught to ensure proper error handling. Overall, these error handling techniques ensure the proper functioning of the project and help avoid any unexpected errors.

List of error handling as follows:

- Command line error
- IllegalArgumentException
- IOException
- FileNotFoundException
- NumberFormatException
- SocketException
- UnknownHostException

D. Critical analysis of the GUI

The user interface of the project was designed to create a user-friendly interface that would allow users to easily navigate the different features of the application and provide helpful feedback in case of errors or unexpected behavior. This section will discuss the key design choices that be made and how they contributed to the overall excellence of our project.

1. MainFrame

The MainFrame, as the centerpiece of our user interface, provides a simple and intuitive means for users to interact with the application's features. Users can easily input their desired word to be searched, added, removed or updated through a dedicated text box. The four buttons provided, namely "Search", "Add New Word", "Remove Word" and "Update Meaning", have been designed with clear and concise labels that give users a clear understanding of the intended function of each button. This design choice ensures that users can quickly and efficiently perform the desired action on their selected word (refer to Appendix B Figure 6).

2. Add New Word

If a user selects the "Add New Word" option, the MainFrame class would check whether the target word is already in the dictionary, which is stored in the HashMap of the Dictionary object.
If the word is already in the dictionary, a pop-up window appears, informing the user that the target word already exists and cannot be added again. To prevent user confusion, the setModal() method has been set to true for this window, which means that the user cannot take any action until they press the OK button (refer to Appendix B Figure 7).
Typically, users will add words that do not yet exist in the dictionary. In such cases, a window for adding the definition of the word appears, allowing the user to enter one or multiple definitions until they press the "Back" button if there are multiple meanings for the word (refer to Appendix B Figure 8, 9).

It is not possible to add a word without any definition. If a user attempts to do so, a window appears, prompting them to either go back to the add definition window to add a definition or delete the meaningless word by pressing the "Delete Word" button, after which the application returns to the MainFrame (refer to Appendix B Figure 10, 11).

3. Search

Searching for definitions is the primary function of the dictionary application. By pressing the "Search" button, users can quickly look up the meaning of the desired word. If the word exists in the dictionary, the definition window will display the word along with its definition(s), each with a corresponding number to distinguish them (refer to Appendix B Figure 12).

4. Update meaning

Clients have the ability to update the meaning of a target word by typing it in the text field and pressing the “Update Meaning” button. If the word exists in the dictionary, the add definition window will appear, just like when a user tries to add a new word. It is important to note that once a user decides to update the meaning of a target word, all of the meanings stored in the dictionary for that word will be deleted. A warning window will appear, prompting the user to press “OK” to update the meaning or “Cancel” to go back to the MainFrame in case they regret or accidentally pressed the button (refer to Appendix B Figure 13, 14, 15).

5. Remove Word

Users can remove existing words from the dictionary by clicking the “Remove Word” button. When the button is pressed, the application first checks if the word exists in the dictionary. If the word is found, it is removed from the dictionary. Once the removal process is completed, a window pops up to inform users that the target word has been successfully removed from the dictionary. The window is setModal() to ensure that users finish with the window before proceeding. After the user presses the “Back” button, the application returns to the MainFrame (refer to Appendix B Figure 16).

6. No Word Found Error

If the target word is not present in the dictionary, the user will not be able to perform any action related to that word, such as searching, updating or removing it. To address this, a window will pop up informing the user that the word is not in the dictionary. To ensure the user acknowledges the message before proceeding, this window is setModal() to true. Once the user clicks "OK" and disposes of the prompt window, they can then proceed to perform their next action in the MainFrame (refer to Appendix B Figure 17).

7. User input blank Error

If the user happens to leave the text field blank before taking any action, all of the buttons will become invalid and pressing them will have no effect (refer to Appendix B Figure 18).

8. Advantages and Disadvantages

The design of the user interface in this dictionary application was carefully considered to provide a user-friendly experience that enables users to easily navigate the different features of the application. While there are several advantages to the design choices that have been made, there are also some potential disadvantages that could be improved or be aware of. This section will discuss the advantages and disadvantages of the UI design.

Advantages:

- User-friendly: The MainFrame provides a simple and intuitive means for users to interact with the application's features.
- Clear Label: The clear and concise labels of the buttons give users a clear understanding of the intended function of each button, allowing them to quickly and efficiently perform the desired action.
- Feedback and Caution: The pop-up windows that appear in response to user actions provide helpful feedback and guidance, preventing user confusion and errors.
- setModal() feature: The setModal() method used for some windows ensures that users cannot take any action until they address the prompt, which helps to prevent accidental or unwanted actions.
- Error Handling: The No Word Found Error and User Input Blank Error handle situations when the user inputs a nonexistent word or leaves the input field blank.

Disadvantages:

- Lack of keyboard shortcuts: The UI does not have keyboard shortcuts, which could slow down users who prefer to use the keyboard instead of the mouse.

- No warning before removing a word: The current design does not provide a confirmation dialog box or any other type of warning before deleting a word. This could potentially lead to the accidental removal of important words from the user's dictionary.

- No keyword menu: A keyword menu could improve the user experience by helping users locate specific words they are searching for, particularly if they are unsure of the spelling or if the word is less common. A keyword menu would present a list of existing words related to the search term, saving time for users and reducing the likelihood of typos or errors.

E. Critical analysis of the thread architecture

The multithreading architecture utilized in this project is the Thread-per-connection architecture. This design assigns a dedicated thread to each client connection, allowing for the simultaneous processing of multiple requests from different clients. This approach is particularly suitable for applications such as a dictionary, where each client's requests are independent and may involve multiple requests, such as searching for multiple words.

The main advantage of this architecture is its simplicity of implementation, which makes it easy to understand and maintain. Additionally, it ensures that each client's requests are isolated from others, preventing one client's request from blocking or affecting the others. This is important in applications where multiple clients are accessing the server concurrently, as it prevents a single client from monopolizing the resources and delaying other clients' requests.

Furthermore, the Thread-per-connection architecture performs well when the number of connections is low. This is because each thread can handle a client's request immediately without waiting for an available worker thread. This allows the server to efficiently utilize its resources, ensuring that requests are processed quickly and efficiently.

However, this approach has some drawbacks. The most significant disadvantage is that creating a new thread for each connection can be resource-intensive, especially for applications with a large number of clients. This can lead to increased memory usage and can degrade the overall performance of the application.

Appendix A.

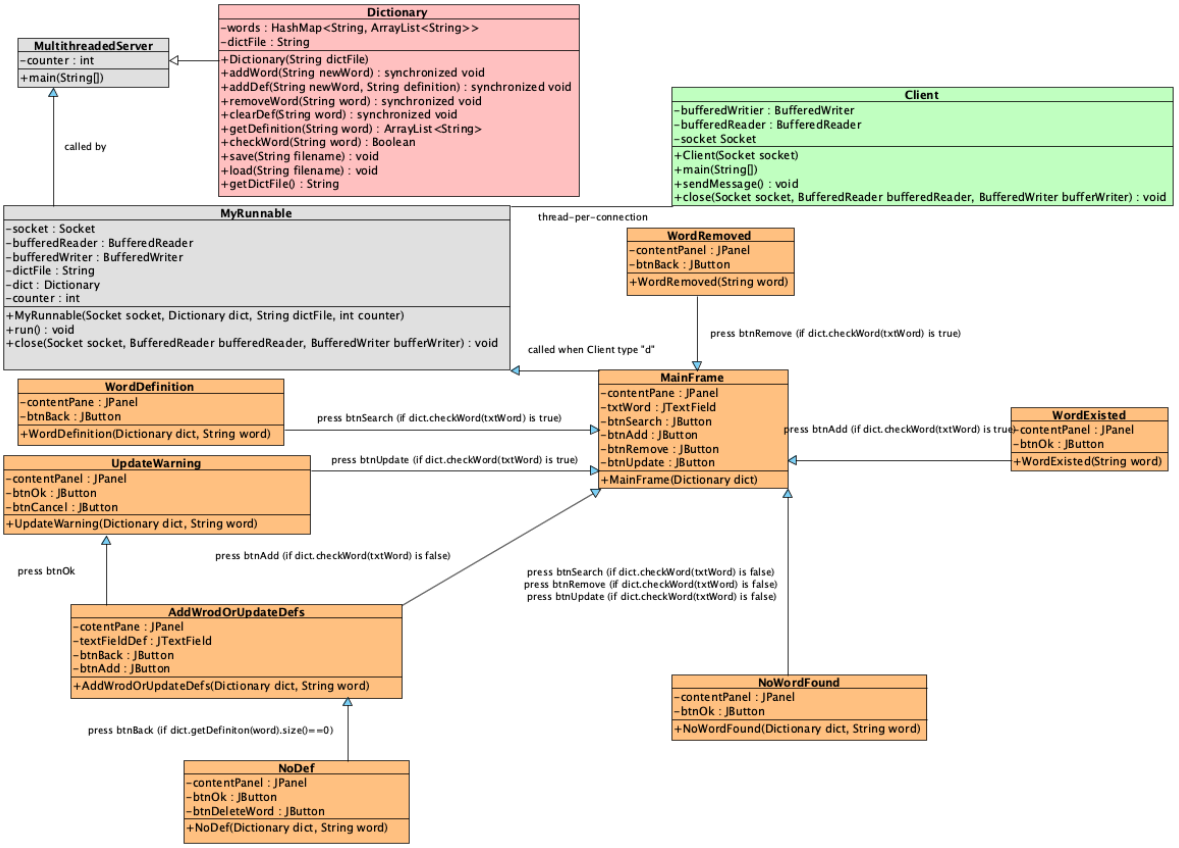

***Figure 5**. UML Diagram - Overview*

Appendix B.

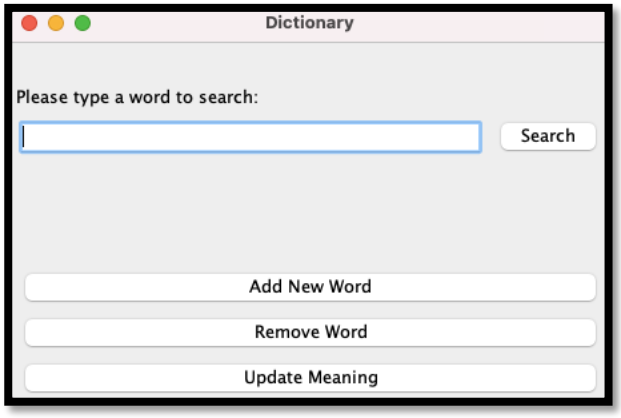

***Figure 6**. MainFrame Window*

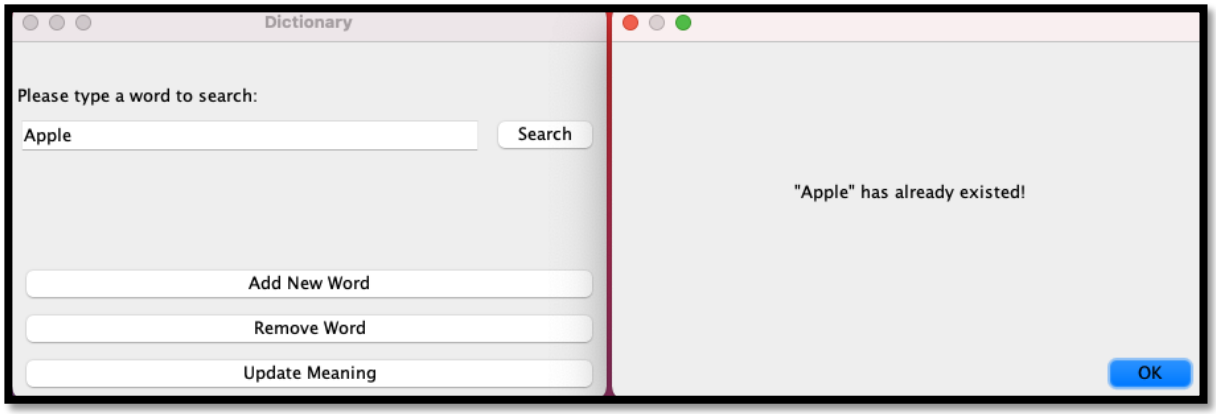

Figure 7. *Add word while the word has existed already*

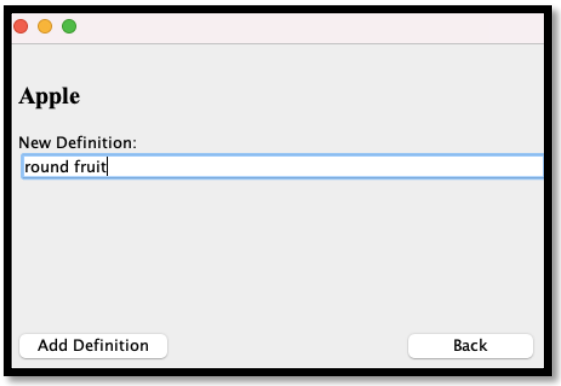

Figure 8*. Add a definition to the word*

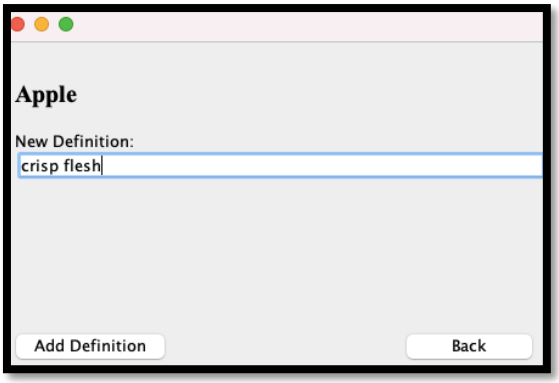

Figure 9. *Add another definiton to the word*

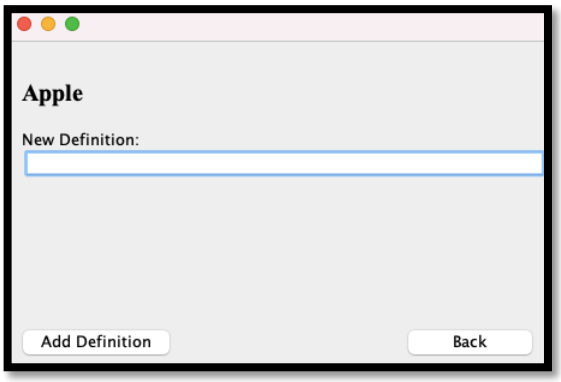

Figure 10. *No definition added to the new word*

Figure 11. *Window warning user to add definition or delete the target word*

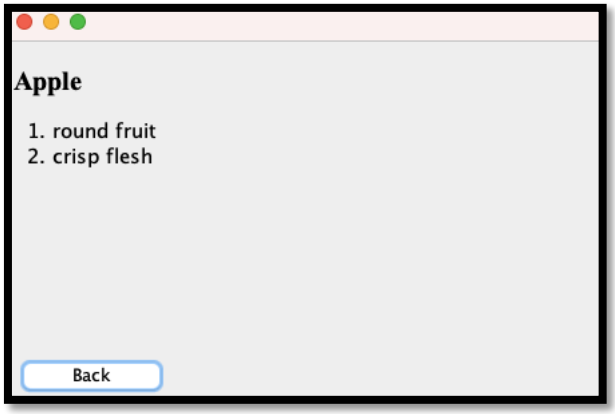

Figure 12. *Search for new word "Apple"*

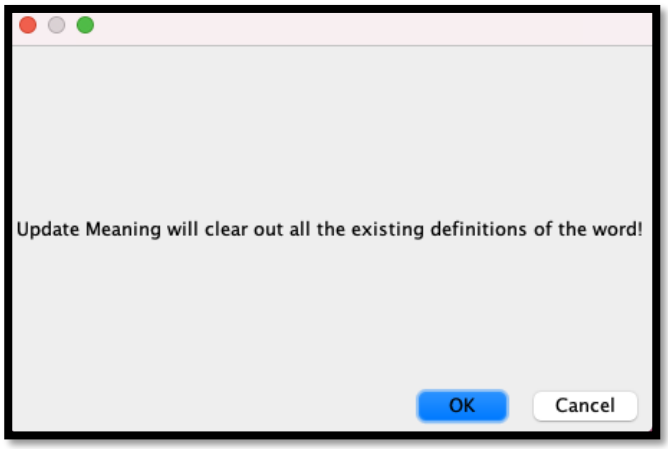

Figure 13. *Warning window when user try to update the meaning of a word*

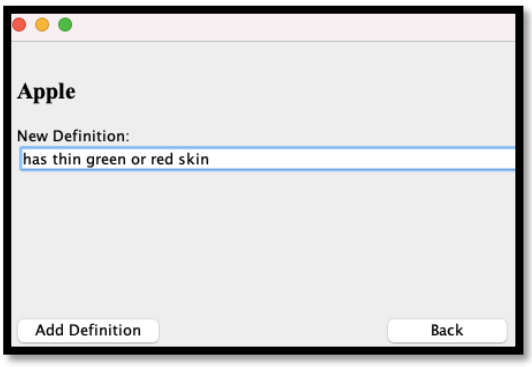

Figure 14. *Insert new meaning to word "Apple"*

Figure 15. *The definition of "Apple" has been updated*

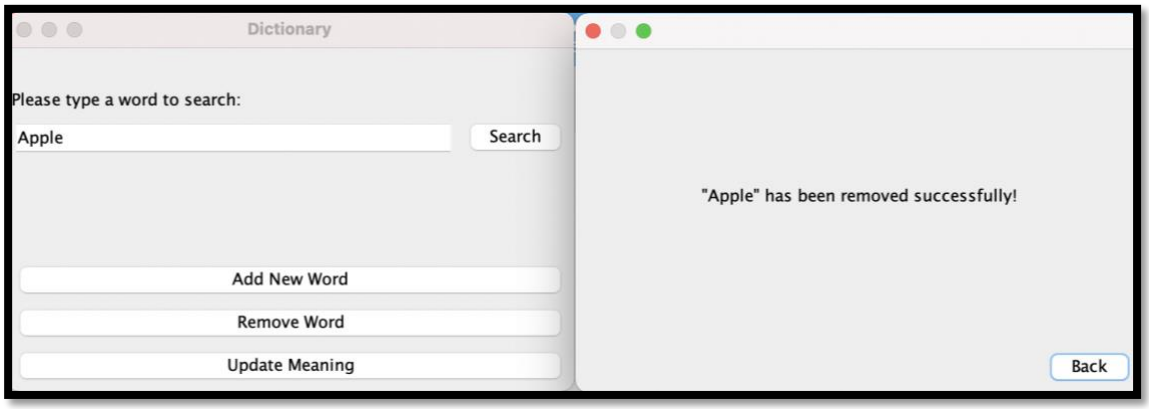

Figure 16. *Remove "Apple" from dictionary*

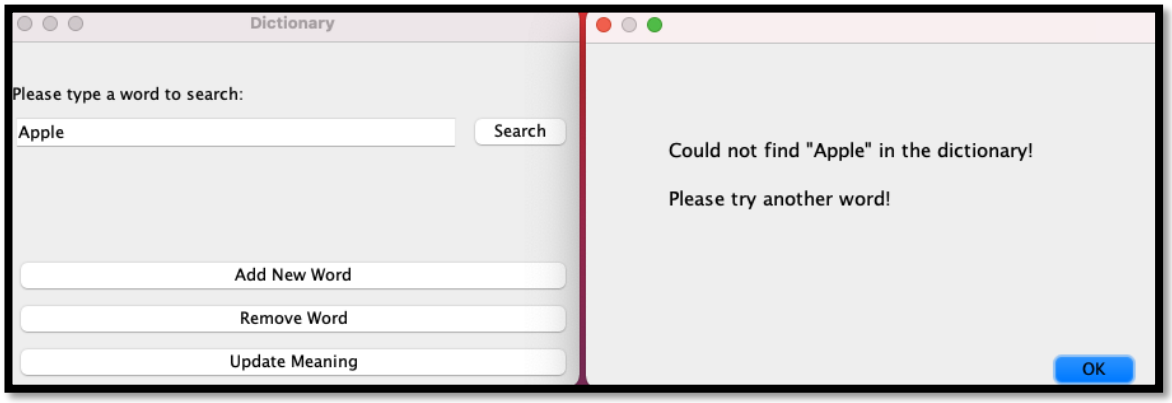

Figure 17. *Target word not found in dictionary*

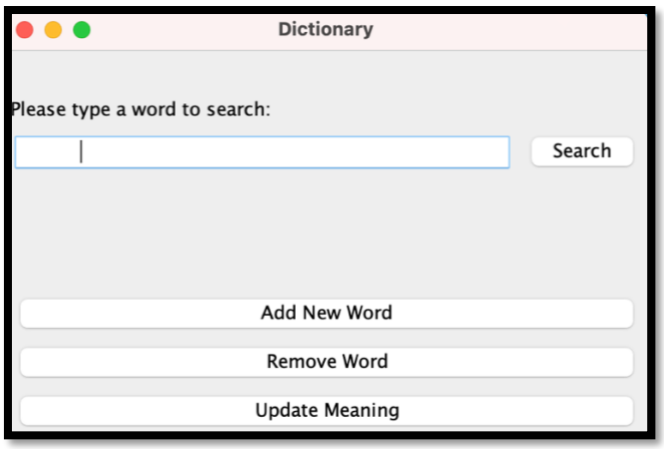

Figure 18. *All of the button invalid when user input blank or leave it empty*