Assignment - 1

Multi-threaded Dictionary Server

Name: Vighnesh
Surname: Chenthil Kumar
Student ID: 1103842

Problem statement

This assignment requires to use a client-server architecture to design and implement a multi-threaded server that allows simultaneous clients to query the meaning(s) of an existing word, add a new word and its meaning, as well as delete any existing word.

Description of Components

The two main components in the system are:

1. **Multi-threaded Server**: The server opens a server socket at first, and listens for connection requests from **all** potential clients through a given port.
   The server then proceeds to read and parse the **dictionary JSON file** into a parsable **JSONobject**.
   As it receives each connection request from the clients, it accepts and establishes a connection between the requesting client and itself, through a separate thread using the **thread-per-connection** architecture.
   Within each thread (server-client connection), it continues to communicate with the client through suitable message passing in the form of **strings**.
   Once the client disconnects itself from the server, the corresponding socket and thread is allowed to close, so that the resources can be freed for another connection.

2. **Client**: The client sends a connection request to the listening port of the server socket, and establishes a connection if accepted by the server.
   As soon as the client-server connection is up and running, a Graphical User Interface (GUI) opens up on the client side to aid the end users with using the application effectively.
   The GUI provides buttons for actions like querying the meaning(s) of an existing word (**Search**), adding a new word and its meaning (**Add**), as well as deleting any existing word (**Delete**).
   On clicking the **Quit** button, the connection is disconnected and the application (GUI) closes on client side.

Running the Application

1. To run the application on the server side, enter the following on the command line:

```
java -jar DictionaryServer.jar <port> <dictionary-file>
```

here, <port> is the port number where the server will listen for incoming client connections, and <dictionary-file> is the path to the file containing the dictionary data.

2. To run the application on the client side, enter the following on the command line:

    java -jar DictionaryClient.jar <server-IP-address> <server-port>

here, <server-IP-address> is the IP address of the server, and <server-port> is the port number where the server will listen for incoming client connections.

Note: An error will be thrown if more (or less) than 3 arguments are input as command-line arguments, or if an invalid dictionary file/IP address/server port number have been input.
In each case, the application will be shut down on either the server or client side (depending on which side the faulty argument was input), as the application will cease to connect without these essential inputs.

Excellence Elements

(i) The following are the two main **design choices** that I made over the regular client-server multi-threading architecture:

1. **Thread-per-connection architecture**: The server creates a new thread for communication every time a new client connects to it.
   This choice was made to ensure that each client gets to interact one-on-one with the server, and does not have to wait at all, especially if its pending request is an urgent one.
   For example, if a client observes that a word has a wrong meaning stored in the dictionary, then it needs to make the suitable changes before the other clients can access the same word, and it cannot afford to be kept in waiting at such a time.

2. **Synchronized** functionality to access the common dictionary object: Here, the server locks the dictionary object each time a client wants to access it, no matter what the nature of the access may be.
   For example, if Client A and Client B want to search and delete a word from the dictionary respectively at the same time, then we need to ensure that either Client A can search for the meaning of the word before it is deleted by Client B, or it shouldn't be able to access the word in the first place as it would have already been deleted by Client B.

(ii) For **failure models**, an error/exception will be thrown if:

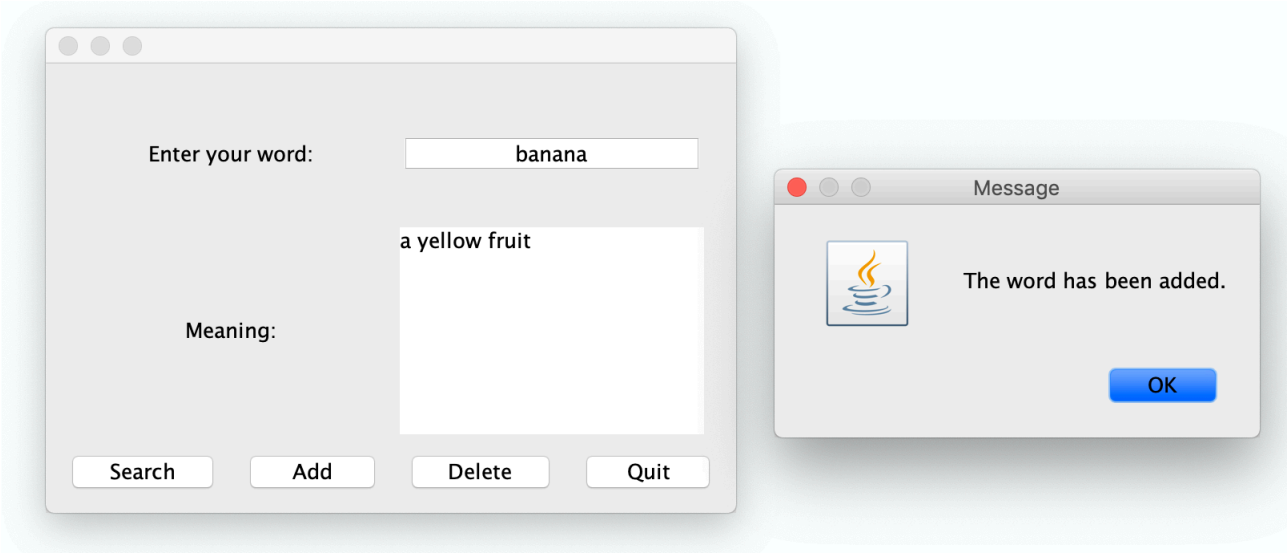
1. more (or less) than 2 command-line arguments are entered
2. an invalid server IP address/server port number/dictionary file is entered
3. unable to parse the dictionary JSON file due to a variety of reasons like corrupted file, wrong file structure etc.
4. unable to create a server/client socket
5. unable to establish/maintain a server-client connection (and communication) through threads
6. the client is run before the server
7. the server gets closed before the client

(iii) For **error notification**, a pop-up message appears on the client-side GUI if:

1. a user tries to search/add/delete an invalid word that contains any non-alphabetic characters, or even blank words
   Note: Leading and trailing whitespaces have been handled internally, and only the word is sent to the server, without the need for any error pop-up messages.
2. a user to tries to search for the meaning of a word that does not exist in the dictionary
3. a user tries to add a word without entering its meaning
4. a user tries to add a word that already exists in the dictionary
5. a user tries to delete a word that does not exist in the dictionary

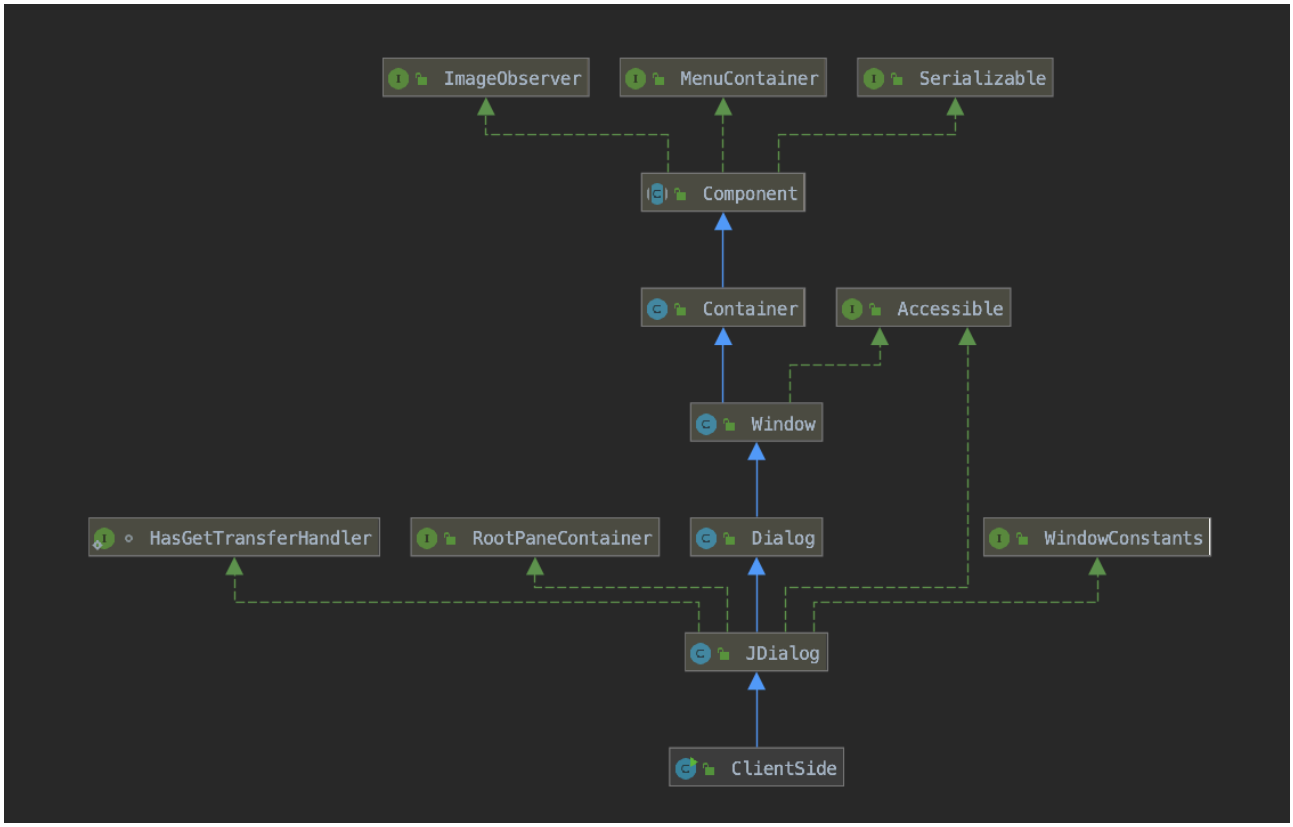
Creativity Elements

1. The client-side of the application uses a Graphical User Interface (GUI) with **Search**, **Add**, **Delete**, and **Quit** buttons for the respective actions on the dictionary, so as to ease the interaction between the client and the server. In addition, there are pop-up messages to notify the user of any errors/updates in their interaction.

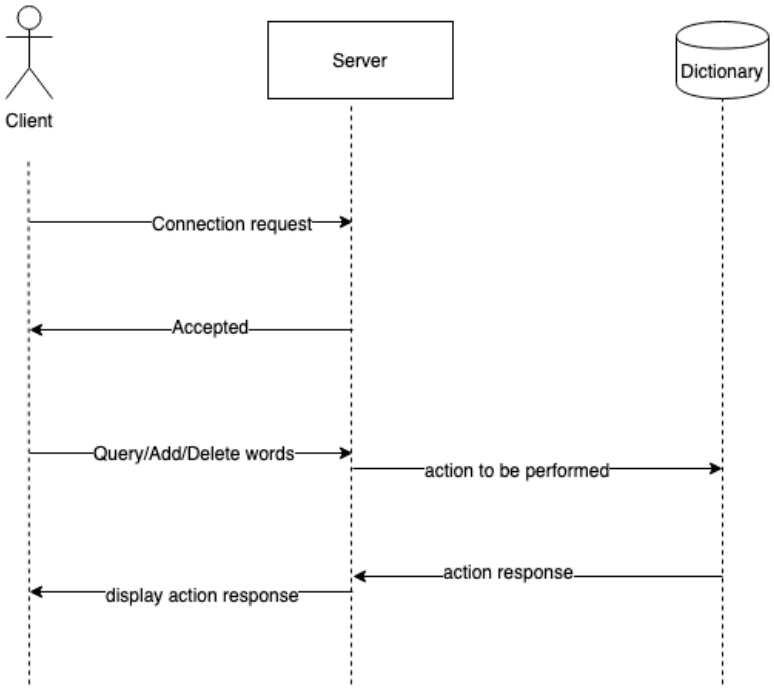


2. The thread-per-connection architecture has been implemented on my own as well, without using any inbuilt Java implementation.

Class Design



Interaction Diagram



Critical Analysis

An analysis can be done on the two main design choices that I have considered as part of this assignment.
Note: Since the advantages of these design choices have already been described under the **Excellence Elements** section, I will be analysing the possible disadvantages/ improvements here.

1. **Thread-per-connection architecture**: Since the Operating System only has a limited number of threads to offer to the server, if a large number of clients were to send connection requests to the server, only those many connections would be accepted as the number of threads in store.
   In this case, even if some later client with a more urgent need to access the server were to send a connection request, it might be kept waiting in place of a more casual client.

2. **Synchronized** functionality to access the common dictionary object: Since search is an operation which doesn't modify the dictionary object, we could use an alternate functionality called read/write locks to allow the client to read (search for meanings of words) even while a write (add/delete) operation is ongoing by another client.
   In case of synchronised, the entire dictionary object is placed under a lock from all other users for read as well as write access.