DISTRIBUTED SYSTEMS (COMP90015)

NAME: ABDUL HANNAN GHAFOOR

STUDENT ID: 1136001

Semester 1, 2022

ASSIGNMENT 1

MULTITHREADED DICTIONARY SERVER

Context

This assignment requires us to develop a multithreaded Dictionary server along with the GUI. The server, being multithreaded, can support multiple concurrent clients and is able to handle requests at the same time. A GUI based client application is also required which can connect with the server and provide some operations. Server supports multiple operations like searching words, adding words and definitions of the words, updating word's definition and deletion of words and their meanings. The server is designed in a way to handle same requests from different clients at the same time without raising clashes. The server and client are both equipped with any failures that may happen during the course of their operation. A custom protocol has also been setup to establish communication between the client and server.

Components.

My project has three components.

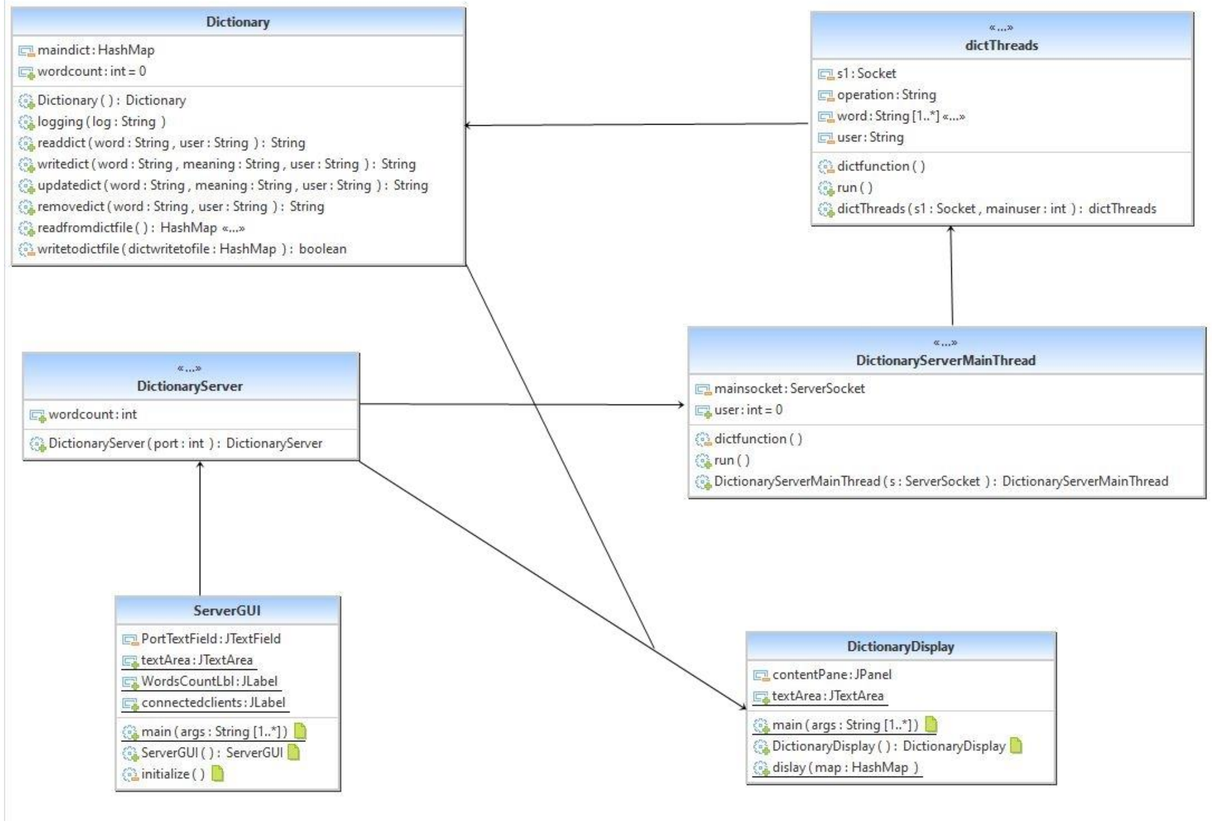
1. Multithreaded Dictionary Server Application (GUI) that provides multithreading per connection (threads per connection)
2. Dictionary Client Application (GUI)
3. Dictionary (File and GUI).

GUI based Server Application first launches the Server's GUI window. That window asks for the port to start the server on. Once it receives the port, it starts the server and launches a main thread on which the server listens. This means that the server application can be extended to include multiple servers features to launch different servers on different ports at the same time from the same GUI. That main thread listens for connections. Once it receives a connection request from a client, it processes that connection request and launches a child thread that is dedicated for that connection. The client then can perform different operations which are.

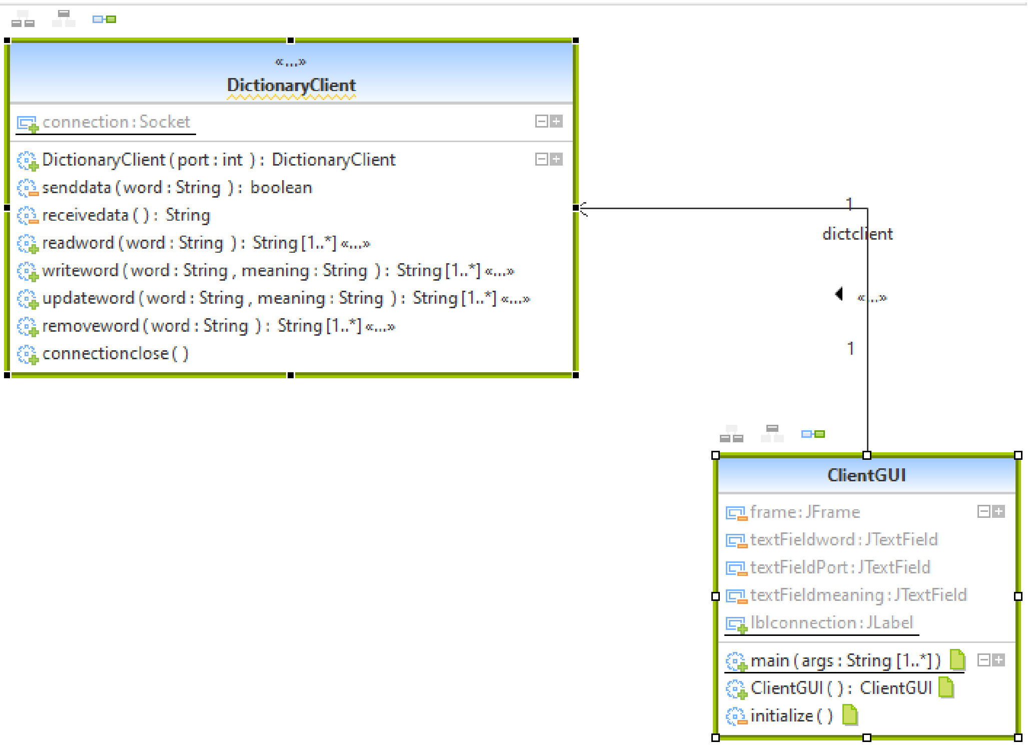
1. Search the word
2. Add a new word and its meaning to the dictionary
3. Add a new meaning to a word that is already available in the dictionary.
4. Update the meanings of the word.
5. Delete the word.

Each operation is followed by saving the changes to the file. This is done so that every client always has the most up-to-date information even when that information that is just recently added or updated. The protocol of communication is implemented using Strings. All the operations and the data are passed through strings to the server and the server sends the status of the operation and the result through the strings as well. The operation and the data are sent as a single string to eliminate the mismatch of operations and data. Both, server and client, are programmed to segregate the control part and data part from the strings. Finally, when the clients want to disconnect, it can just close its application which lets the server know that client has closed the connection, so server also closes the connections from its side and deletes the thread that it had associated with the client.

Class Design:


Dictionary
maindict : HashMap
wordcount : int = 0
Dictionary ( ) : Dictionary
logging (log : String)
readdict (word : String , user : String) : String
writedict (word : String , meaning : String , user : String) : String
updatedict (word : String , meaning : String , user : String) : String
removedict (word : String , user : String) : String
readfromdictfile ( ) : HashMap «...»
writetodictfile (dictwritetofile : HashMap) : boolean
«...»
dictThreads
s1 : Socket
operation : String
word : String [1..*] «...»
user : String
dictfunction ( )
run ( )
dictThreads (s1 : Socket , mainuser : int) : dictThreads
«...»
DictionaryServer
wordcount : int
DictionaryServer (port : int) : DictionaryServer
«...»
DictionaryServerMainThread
mainsocket : ServerSocket
user : int = 0
dictfunction ( )
run ( )
DictionaryServerMainThread (s : ServerSocket) : DictionaryServerMainThread
ServerGUI
PortTextField : JTextField
textArea : JTextArea
WordsCountLbl : JLabel
connectedclients : JLabel
main (args : String [1..*])
ServerGUI ( ) : ServerGUI
initialize ( )
DictionaryDisplay
contentPane : JPanel
textArea : JTextArea
main (args : String [1..*])
DictionaryDisplay ( ) : DictionaryDisplay
dislay (map : HashMap)

The ServerGUI is started up first. It has the main function and all the ServerGUI related code. Upon pressing the Connect button, an object of DictionaryServer class is created. DictionaryServer then spins up object of DictionaryDisplay class which creates the GUI for the dictionary. DictionaryServer class also creates the object of DictionaryServerMainThread. DictionaryServerMainThread creates the serversocket and listens for connection requests. Once client tries to connect with the Server it then creates the object of dictthreads class and create a seprate thread for that client. Dicthreads class then parse the requests according to the protocol mentioned below and performs each operations through the object of dictionary class. Dictionary class has all the methods for the allowed operations as well as the methods to read and write from the dictionary file. DictionaryServer class also use dictionary class to initialize the dictionary.


«...»
DictionaryClient
connection : Socket
DictionaryClient (port : int) : DictionaryClient
senddata (word : String) : boolean
receivedata ( ) : String
readword (word : String) : String [1..*] «...»
writeword (word : String , meaning : String) : String [1..*] «...»
updateword (word : String , meaning : String) : String [1..*] «...»
removeword (word : String) : String [1..*] «...»
connectionclose ( )
1
dictclient
«...»
1
ClientGUI
frame : JFrame
textFieldword : JTextField
textFieldPort : JTextField
textFieldmeaning : JTextField
lblconnection : JLabel
main (args : String [1..*])
ClientGUI ( ) : ClientGUI
initialize ( )

Above UML diagram shows the class structure for the client. ClientGUI class has the main function. This class calls the functions from the DictionaryClient, which has all the operation functions along with the sending and receiving data function, when the operator buttons are clicked on the client GUI application.

Protocol:

Following protocol has been implemented between the client and the server.

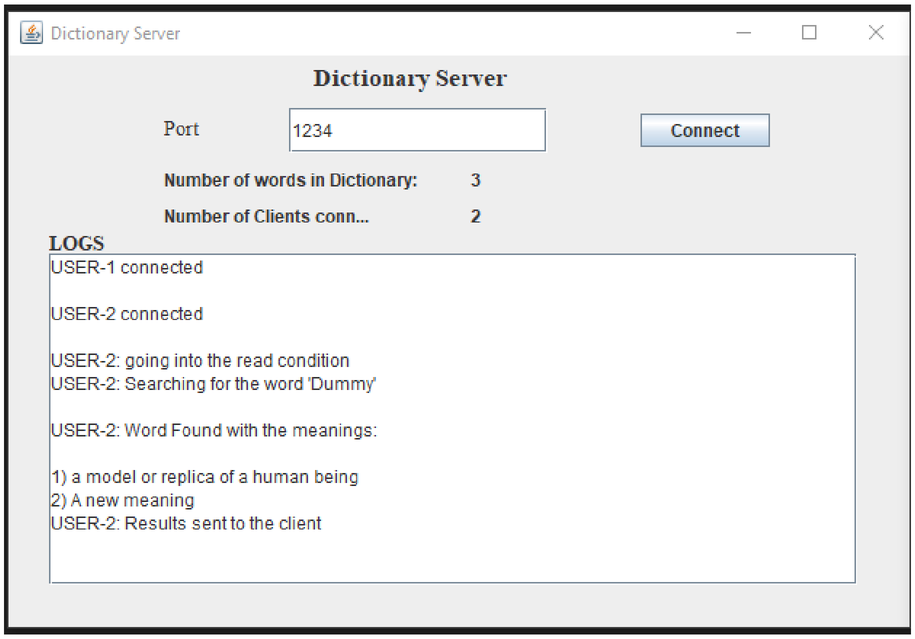
- For Add operation:
  - Client - > Server: “write@Word@Meaning”
  - Server - > Client:
    - In case of success: status(success)@Meaning1@Meaning2@...MeaningN
    - In case of failure: status(failure)@StatusMessage
- For Delete operation
  - Client-> Server: “delete@Word”
  - Server - > Client:

        - In case of success: status(success)@StatusMessage
        - In case of failure: status(failure)@StatusMessage
- For update operation:
    - Client - > Server: “update@Word@Meaning”
    - Server - > Client:
        - In case of success: status(success)@Meaning1@Meaning2@...MeaningN
        - In case of failure: status(failure)@StatusMessage
- For Search operation:
    - Client -> Server: “read@Word”
    - Server - > Client:
        - In case of success: status(success)@Meaning1@Meaning2@...MeaningN
        - In case of failure: status(failure)@StatusMessage
- For Disconnect operation:
    - Client -> Server: “Disconnect”

GUI

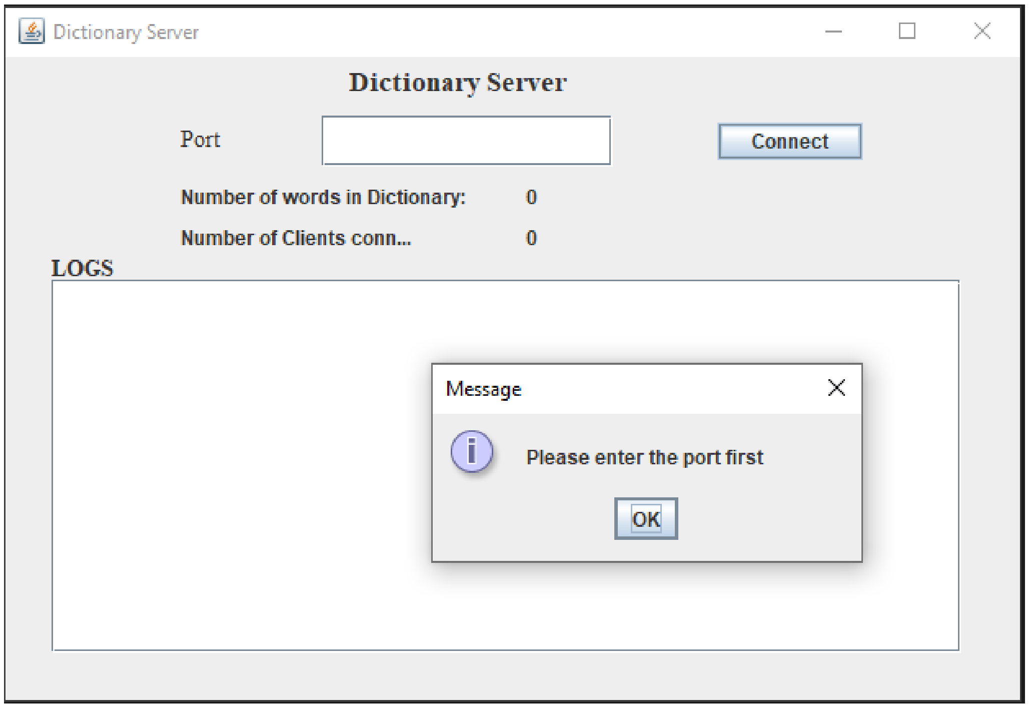
The GUI is designed in the following way.

Server:

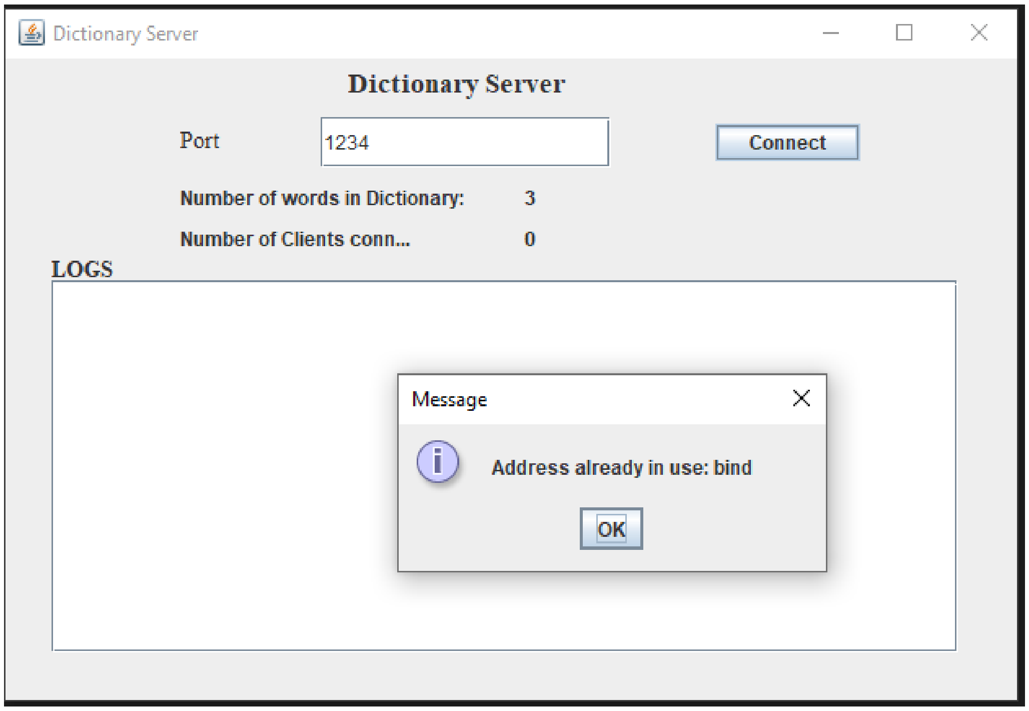


```
USER-1 connected

USER-2 connected

USER-2: going into the read condition
USER-2: Searching for the word 'Dummy'

USER-2: Word Found with the meanings:

1) a model or replica of a human being
2) A new meaning
USER-2: Results sent to the client
```

As shown in the pic, the Server GUI contains a textField to get the port number. There is a connect button to connect to the server running at localhost. There is a logs portion which shows the logs of every user along with the status messages from the server. Important exceptions are shown through the Message box while and shown in the logs as well. The GUI also show the number of words in the dictionary and the number of clients connected. These numbers are updated on a real time basis.



Please enter the port first



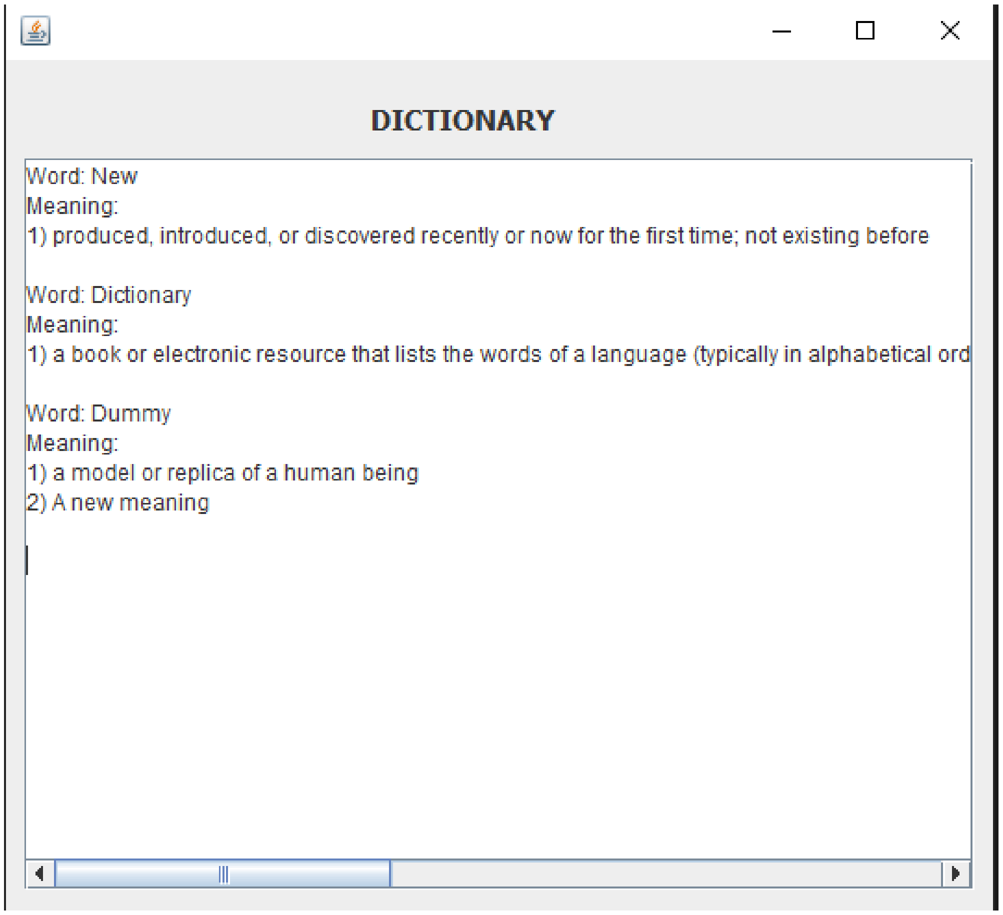
Dictionary Server

Port: 1234

Number of words in Dictionary: 3

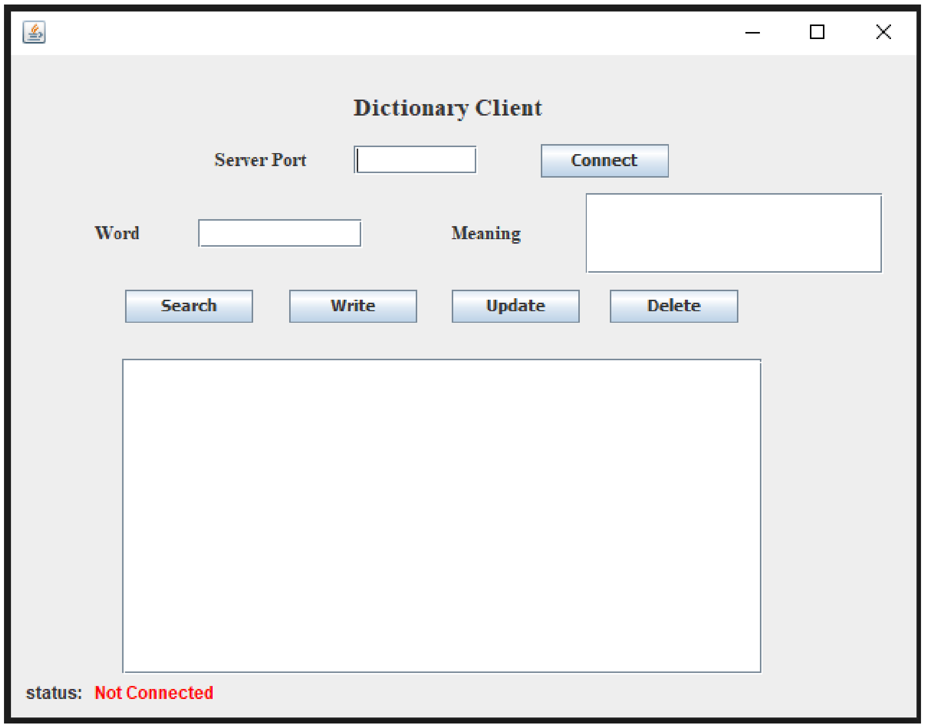
Number of Clients conn... 0

Address already in use: bind



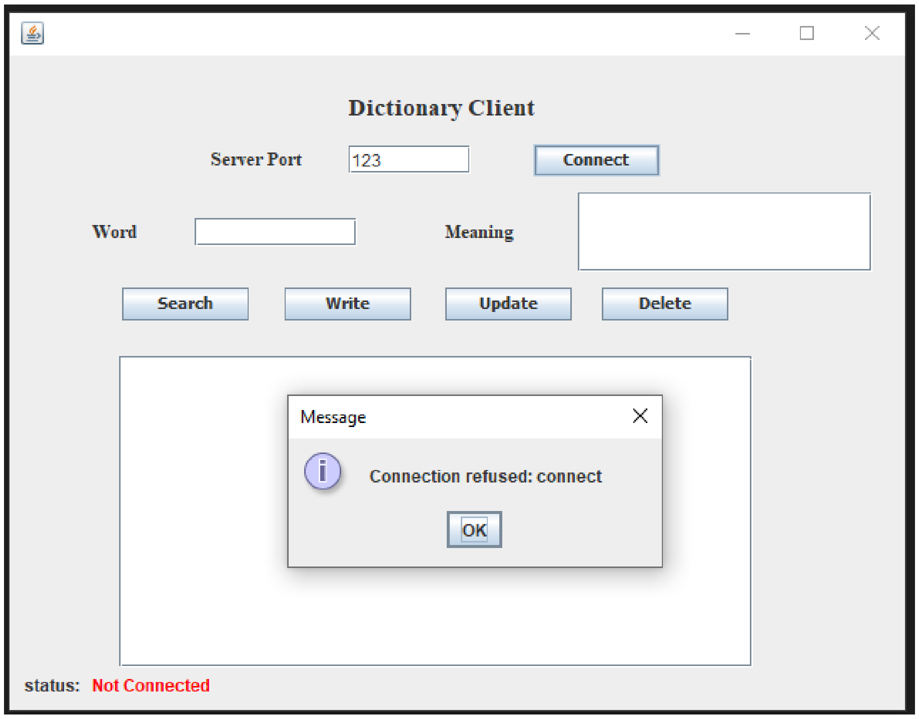
```
Word: New
Meaning:
1) produced, introduced, or discovered recently or now for the first time; not existing before

Word: Dictionary
Meaning:
1) a book or electronic resource that lists the words of a language (typically in alphabetical ord

Word: Dummy
Meaning:
1) a model or replica of a human being
2) A new meaning
```

The server also has a GUI of the Dictionary. All the words and their meanings are displayed over here. This GUI is updated on a real time basis.



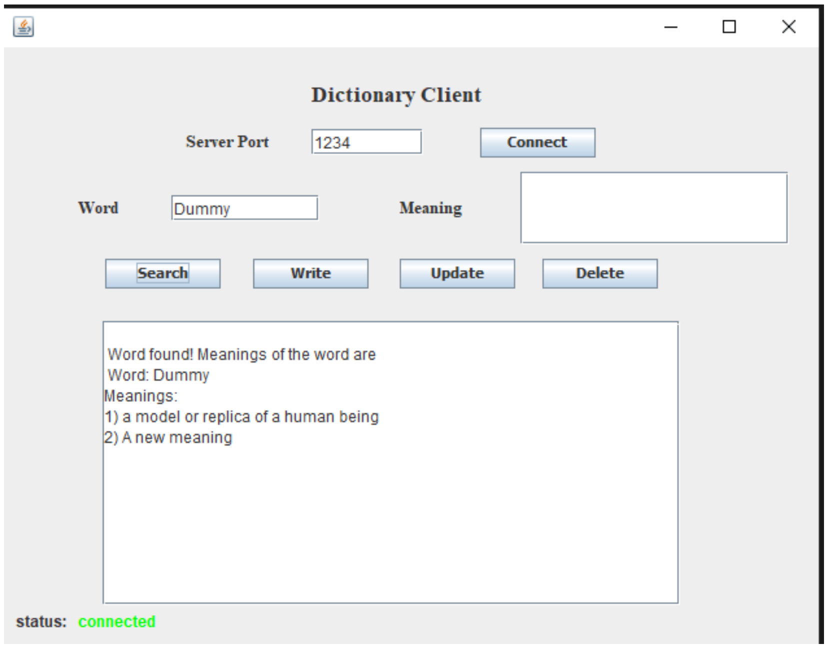
status: Not Connected

Above is the pic of client GUI. It indicates when the client is not connected with the server.



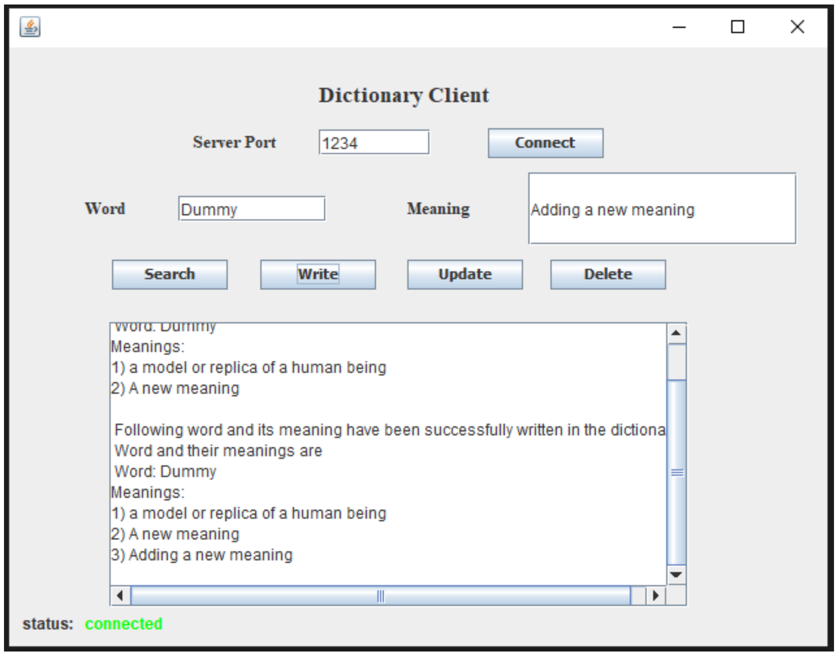
Connection refused: connect

status: Not Connected



```
 Word found! Meanings of the word are
 Word: Dummy
Meanings:
1) a model or replica of a human being
2) A new meaning
```

status: connected



```
Word: Dummy
Meanings:
1) a model or replica of a human being
2) A new meaning

 Following word and its meaning have been successfully written in the dictiona
 Word and their meanings are
 Word: Dummy
Meanings:
1) a model or replica of a human being
2) A new meaning
3) Adding a new meaning
```

status: connected

Above are the results of the search and write operations done through client application. The above diagrams are for reference. They don't show all operations and errors handled by the Server and Client applications.

Critical Analysis:

My Server Application use threads per connections model for multithreading which I think is a superior method to implementing multithreading compared to Threads per Requests. The latter would create a lot of threads if there are a lot of clients connected and destroy them immediately once the server finishes the operations. This makes the thread per request model highly inefficient as there is always an overhead of creating a new thread every time a request comes in. Threads per connections create threads per client connection, which means that there is still concurrency present. Each thread will be dedicated to each client and will be destroyed once the client closes the connection. Server uses HashMap data structure to store the dictionary in memory. It reads the dictionary from the file each time any operation is called, performs the operations and if needed, write the changes in the dictionary files. In order to make sure that only one thread is doing the read and write operations to the dictionary file at a given time, I have created separated readfromdictfile and writetodictfile functions which are synchronized. Writetodictfile function also uses readtodictfile function to ensure that file write and read operations don't happen at the same time. Even though my implementation of writing to dictionary file and reading from dictionary file at every operation is not time efficient (especially when considering 1000s of clients connected with the server) however it ensures that every client will always get the most up-to-date information. I personally, we can neglect this cost considering the benefits that we will have. In my implementation, I am also creating a separate thread for every Server from the same server application. My current GUI does not cater for multiple dictionary servers at the same time however my backend application will work if I ever want to extend my Server GUI to implement multiple server on different ports and addresses.

Creativity:

There are several things that I have done for creativity.

1. Real Time GUI for the Dictionary file which shows all the dictionary on the GUI window. It is launched in a separate window along with the server.
2. All the checks have been imposed to make sure that the dictionary file is created when missing even during the application.
3. Realtime information on the server window about the number of words present in the dictionary and the number of clients connected with the server.
4. Realtime information on the client window when the client is connected with the server.
5. Server logs on the server window which contains all the requests made by the clients. All the operations done by the server in response to those requests and the responses sent to the clients against their requests.
6. Implementation of different threads for each ServerSocket binding so that multiple servers can be initialized on different ports.
7. Dictionary file is updated on a realtime basis to ensure that every client is getting the most updated information.