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

Name: Sanjana Ratan

Student ID: 1041413

1. PROBLEM

1.1 GOAL

The goal of this assignment is to create a multi-threaded server that allows concurrent clients to search the meaning(s) of a word, add a new word with its meaning and delete a word. The implementation follows a client-server architecture where all interactions take place through sockets and threads.

1.2 FAILURES

There can be a number of failures that can possibly occur while interacting with a remote dictionary. The most common and crucial ones are as follows –

- Server goes offline.
- Server is unable to connect with the socket because socket connection is closed.
- Server is unable to set up a socket for an incoming connection.
- Client is unable to communicate with the server.
- Server is unable to parse dictionary file to perform operations.
- Server is unable to receive and send data from and to the client.
- Client socket connection cannot be closed.
- Dictionary file cannot be updated.
- The word to be searched and deleted does not exist in the dictionary.

1.3 CONSTRAINTS

The implementation adheres to a few constraints in context to the problem being solved. These constraints are as follows –

- Sockets and threads must be the lowest level of abstraction for communication and concurrency.
- All communication between client and server must be reliable.
- All errors must be handled efficiently with appropriate notifications at both ends.
- Multiple clients must be able to perform multiple functions concurrently with the same server.

2. ARCHITECTURE

The entire system contains 3 classes in which 1 of them is the core client class that is able to send client requests through a GUI and 2 of them are core server classes that are able to receive and process multiple client requests.

The dictionary file follows JSON format and all interaction with this file is done using two external libraries which are org.json and json-simple-1.1.1. The multi-threaded server follows both worker pool and thread per connection architecture depending on the command line argument passed.

2.1 CLASS DIAGRAM

Dependencies and the definition between the 3 classes are shown with the help of a class diagram –

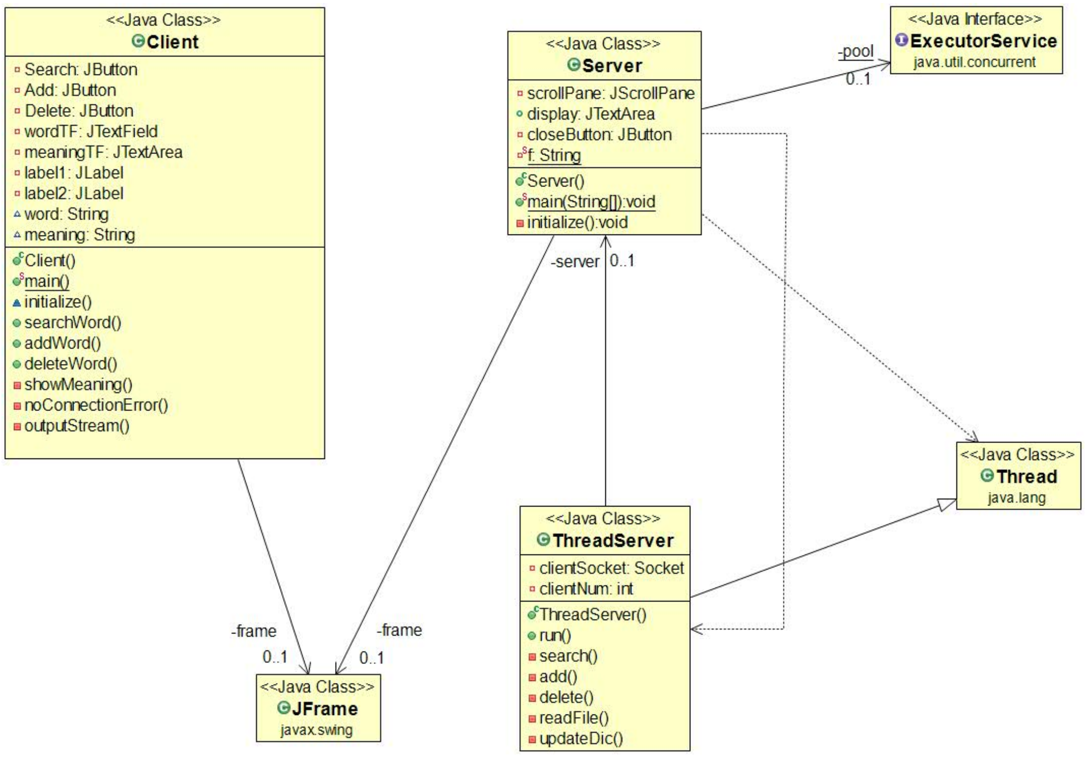


Diagram 1 - Class Diagram

Client is the core class at the client side that uses JFrame to initialize the it's application window. It accepts the word and meanings from client through a text field and text area respectively. The application contains 3 buttons that call 3 separate methods – searchWord(), addWord() and deleteWord(). The client's message is taken from the text field and text area which is passed onto the output stream using the outputStream() method. Server class creates a ServerSocket and accepts incoming client connections, it also uses JFrame to initialize an application window that displays all interaction taking place with the client. This class provides worker pool and thread-per-connection architecture to create a thread for every incoming connection. ThreadServer is the core class that is executed each time a thread is created and it calls 3 different methods depending on the client's message – search(), add() and delete(). This class interacts with the dictionary file by invoking readFile() method and makes changes to it by invoking updateDic() method.

2.2 INTERACTION DIAGRAMS

All the interaction between the classes is shown below –

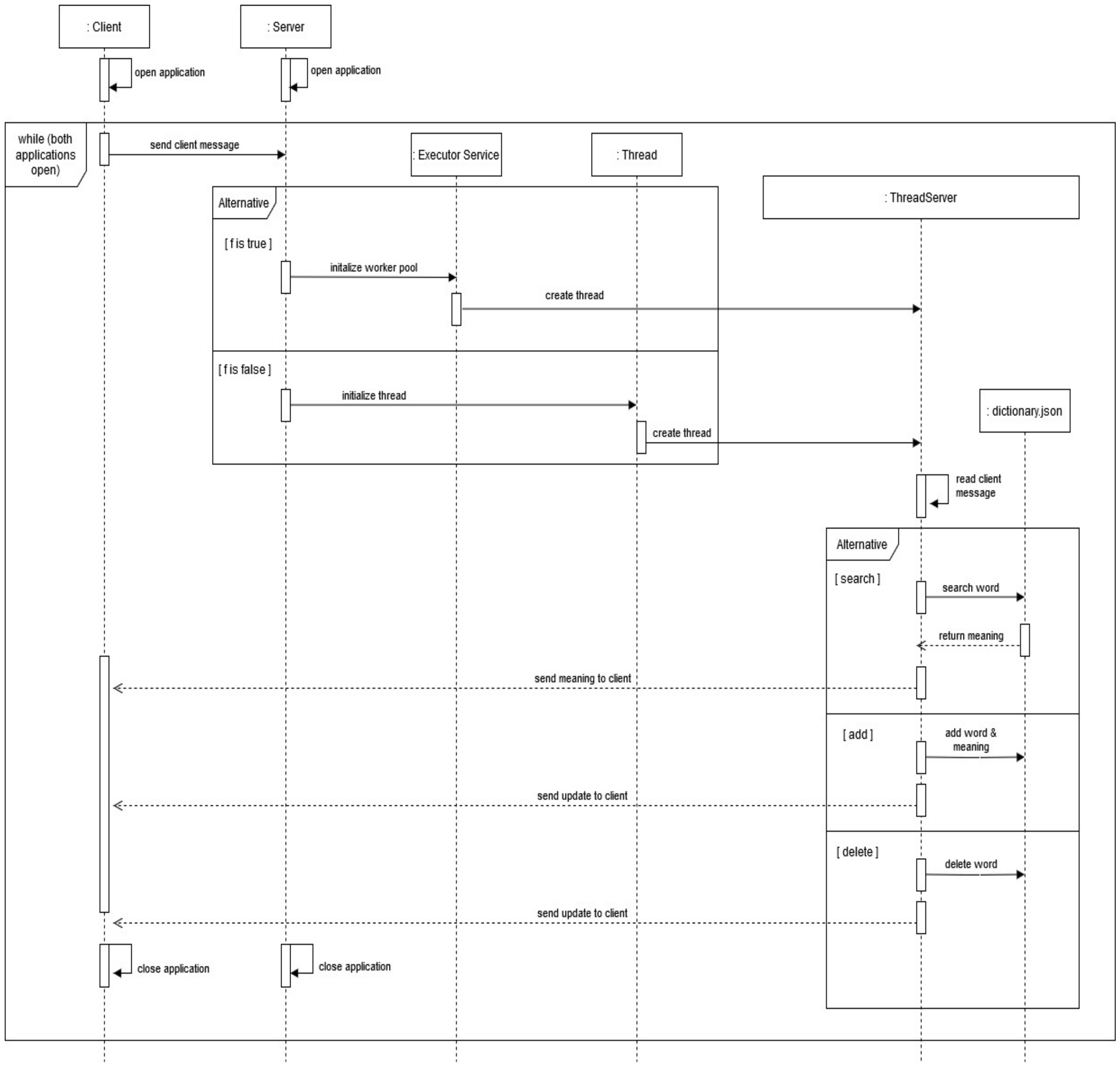


Diagram 2 – System Sequence Diagram

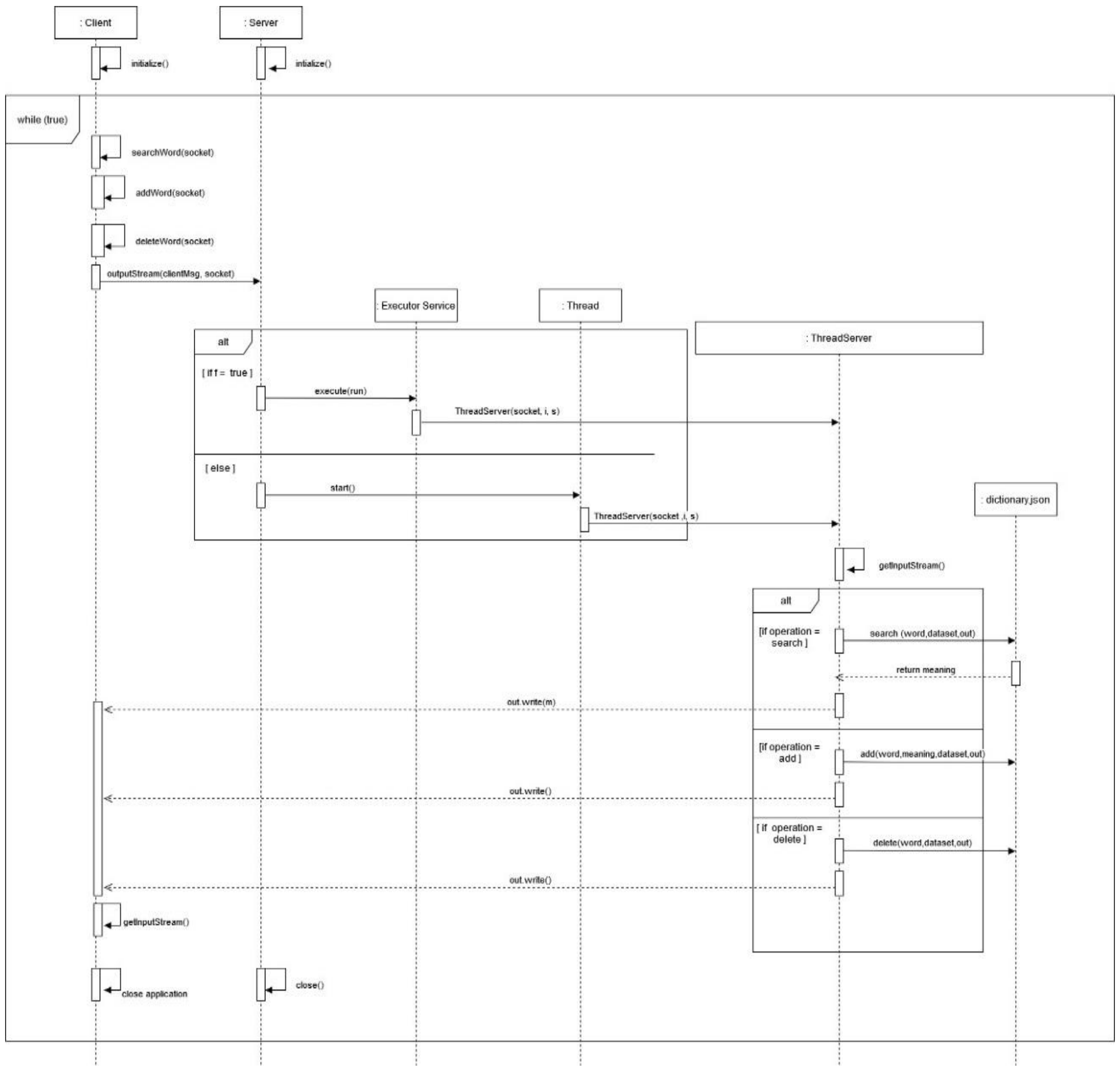


Diagram 3 – Design Sequence Diagram

As long as both applications are running, the clients can make requests through their own thread and the server can make changes to the dictionary file. The design sequence diagram shows the detailed interaction flow between the client and server by including the software artefacts and concepts like methods.

2.3 HOW TO RUN?

To run the jar files please follow the commands below –

- Server – `java -jar Server.jar true/false`

If you wish to use worker pool architecture mention *true* and if you wish to use the thread per connection architecture mention *false*

- Client – `java -jar Client.jar`

3. PROTOCOL

The message protocol used to transmit and read messages back and forth between client and server is very simple. It is sent as a simple string in the following format – `operation word meaning ; .`This is read using the String Tokenizer as it reads each part as a token and each part is stored in variables namely operation, word and meaning. The output from server is sent as a simple string and is read normally on the client side using the input stream.

4. ANALYSIS OF WORK DONE

I have implemented both worker pool and thread-per-connection architecture to be able to analyze each design's advantages and disadvantages when executed. After the implementation and thorough analysis of each design's behavior, the following can be concluded –

Worker Pool	**Thread-per-connection**
1. Allows only a limited number of clients to perform their operations so the dictionary server responds faster.	A thread is created for every connection so the dictionary server has a greater number of clients to process which decreases the response time.
2. It reuses the thread when a client disconnects and another client connects so it saves time rather than creating a new thread for every client connection.	Every time a new thread has to be created and destroyed which is a time-consuming task for the server.
3. There are fixed number of threads so there is no wastage of memory.	The dictionary server has large number of clients and so, memory space has to be allocated to all which leads to wastage of memory.
4. This hangs the client GUI application if more than the allocated number of clients connect.	The client GUI application works smoothly no matter how many clients connect.

5. The client needs to wait if the pool's limit is reached.	The client does not need to wait as a new thread is created immediately and the dictionary server begins processing its request.
6. There is no control over the priority of the thread created for a client.	One can prioritize the threads as per the operations for example, I can prioritize add operation. Therefore, a client with an add word request will be dealt first.
7. The thread is terminated by the executor service and we don't need to terminate it ourselves.	We have to terminate the thread ourselves after a client's connection is closed.
8. In conclusion, this is useful to handle short-duration requests as it works well when the number of clients are limited.	In conclusion, this is useful to handle long-duration requests like in the multi-threaded dictionary server as there can be a large number of clients performing multiple operations at the same time and the server needs to remain online for a longer duration.

5. EXCELLENCE ELEMENTS

The following have been implemented under the excellence section –

I. **Notification of errors** – I have taken a number of errors that can occur during the interaction into consideration and made sure to notify either sides of the error that occurred. Below is a list of ALL the errors taken into account to build an efficient model –

Error	**Handling**
Server is offline	The terminal displays a print message that server has gone offline.
Server is unable to connect with socket	Throws a SocketException and server GUI displays that socket connection has closed.

Server is unable to create socket for incoming connection	IOException is thrown at the time of server.accept() and the GUI displays that there is an error setting up socket.
Client is unable to communicate with server	IOException is thrown and client GUI displays an error window with error description.
Server is unable to parse dictionary file	JSONException is thrown and server GUI displays that server cannot parse the dictionary file.
Server is unable to receive or send to or from client	IOException is thrown and server GUI displays that it is unable to recive/send information.
Client connection cannot be closed	When socket.close() happens, an IOException is thrown and the server GUI displays that it was unable to close the connection.
Dictionary file cannot be updated	While attempting to update the dictionary, an IOException is thrown and the server displays that it is unable to update the dictionary
Word being searched or deleted does not exist in dictionary	The server checks if the dictionary has the word, if not it sends "Word not found" message to client and client GUI displays this message in a pop-up window.

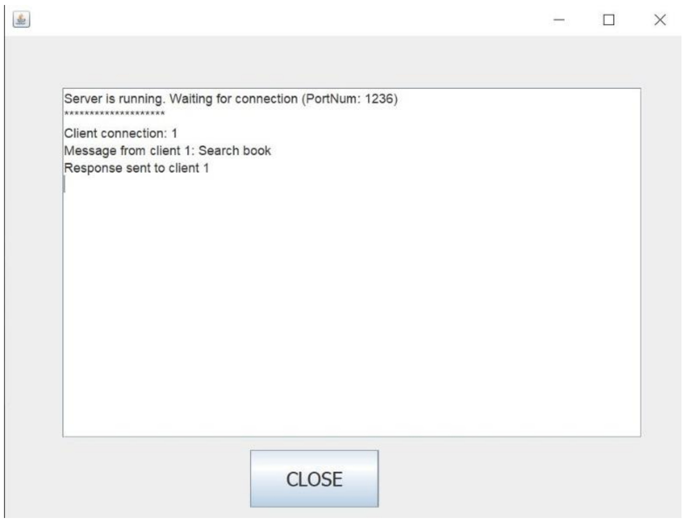
II. **Proper graphs in report** – The report displays a class diagram, system sequence diagram and design sequence diagram to show components and their interaction properly.

III. **Analysis of system including advantages and disadvantages** – *Section 4: Analysis of work done* provides a comparison of both worker pool and thread-per-connection architecture implemented.

6. CREATIVITY ELEMENTS

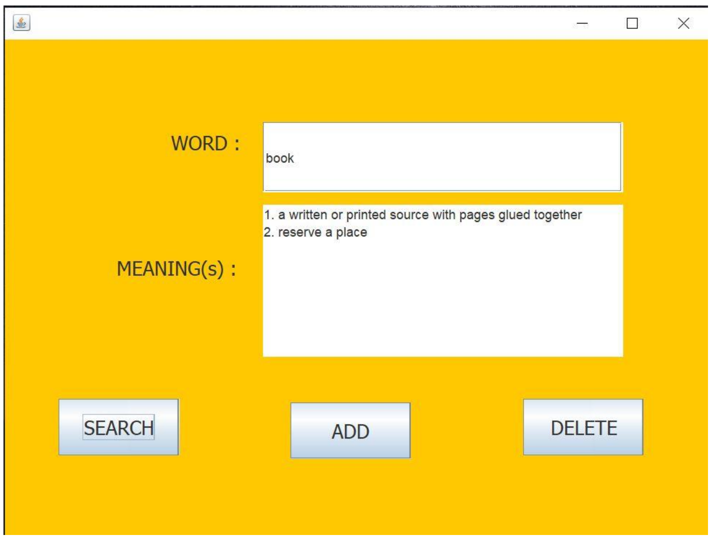
The following have been implemented under the creativity section –

I. **GUI for server** for displaying all interactions with the client.



```
Server is running. Waiting for connection (PortNum: 1236)
********************
Client connection: 1
Message from client 1: Search book
Response sent to client 1
```

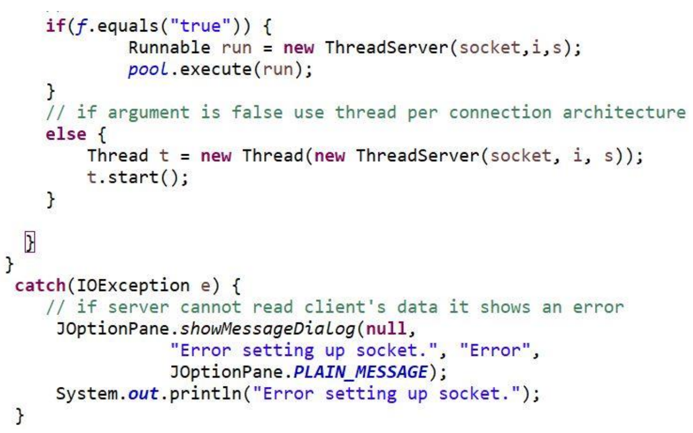
II. Advanced feature of **allowing user to add/search multiple meanings** for the same word.



WORD : book

MEANING(s) :
1. a written or printed source with pages glued together
2. reserve a place

III. **Implementation of two thread architectures** i.e. worker pool and thread-per-connection



```
    if(f.equals("true")) {
            Runnable run = new ThreadServer(socket,i,s);
            pool.execute(run);
    }
    // if argument is false use thread per connection architecture
    else {
        Thread t = new Thread(new ThreadServer(socket, i, s));
        t.start();
    }

  }
}
 catch(IOException e) {
    // if server cannot read client's data it shows an error
     JOptionPane.showMessageDialog(null,
                "Error setting up socket.", "Error",
                JOptionPane.PLAIN_MESSAGE);
     System.out.println("Error setting up socket.");
 }
```