Multi-threaded Dictionary Server

COMP90015: Distributed Systems – Assignment 1

Date of Submission - 5 September 2019

Student Name: Haripriya Ramesh

Student ID: 1073646

Problem Description

The aim of the assignment is to design and implement a multi-threaded dictionary server that allows multiple concurrent clients to query the meaning of a word, to add word and its corresponding meanings, and to delete a word from the dictionary using Java programming language. This should be implemented using client-server architecture with sockets and threads as the lowest level of abstraction for network communication and concurrency. The application will also include a UI component on the client side for interacting with the user.

Architecture

The developed system uses the 'Thread per connection' architecture. A new thread is created by the server to serve each client i.e., a single thread will serve all requests coming from a client. The number of threads created is equal to the number of clients connected to the server. The advantage of using this architecture is that it is simple to implement and is useful when we have fewer clients as there is no need to create many threads. For the dictionary server, the number of clients is not too many so thread per connection is a feasible option.

The communication between the client and server is via the Transmission Control Protocol (TCP) which by nature ensures reliable communication.

System Components

This system has three components, the server component, client component and the dictionary file.

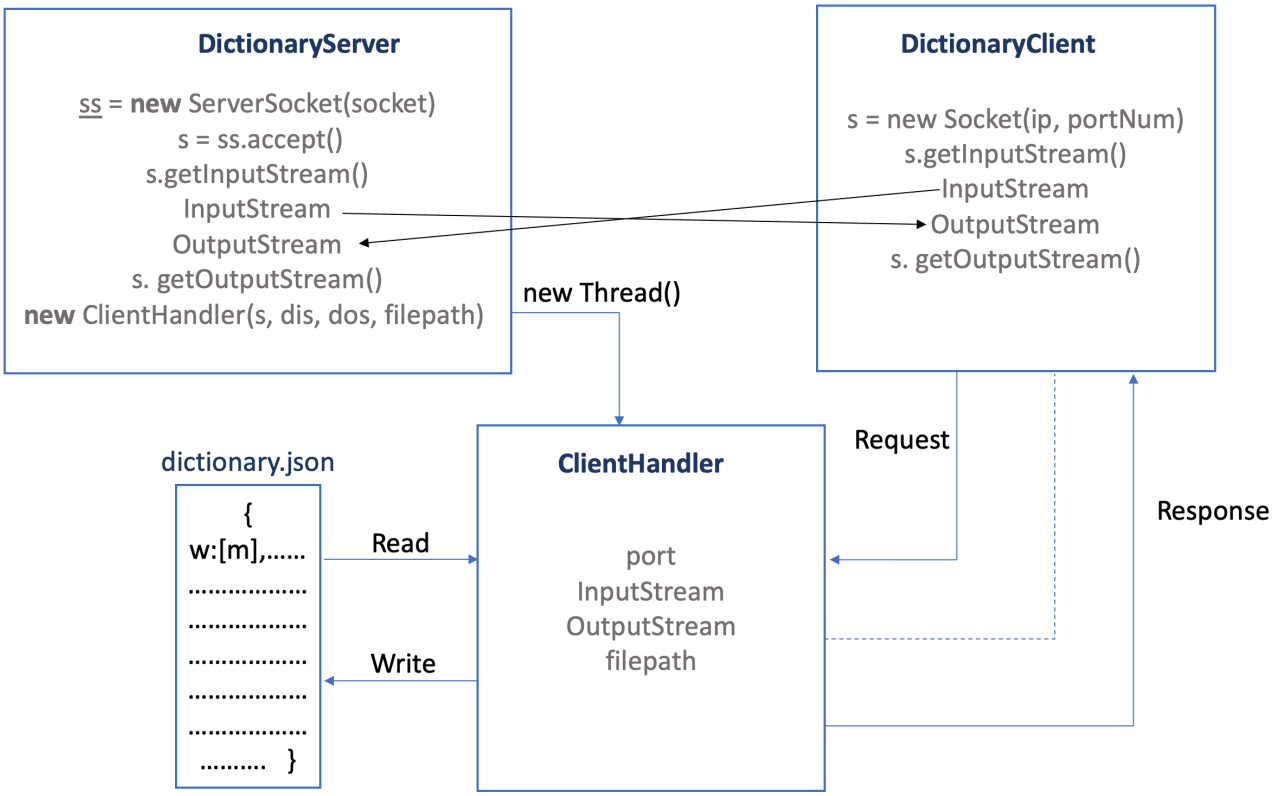

Fig 1: Diagram depicting system components and their interactions.

Server Component:

The server component is divided into two parts.

1. DictionarySever:
    - This is the server class that has the main method.
    - The dictionary server creates sockets for communicating with clients.
    - It receives the clients request to connect.
    - Each client for which the connect request is accepted, it creates a thread.
    - This server is designed to connect to any number of clients until it is manually terminated.
2. ClientHandler
    - This is a part of server component that manages each clients request.
    - The dictionary file is also accessed from this class. It is responsible for performing the operations by reading and writing to the dictionary file.

- The org.json library is used to read and write into the json file. A JSONTokener, JSONObject and JSONArray are used to store the data read from file.
- It processes the request by calling various methods that perform operations to get the response which is then communicated to client.
- It can process multiple requests from each client that is sent one after the other through the GUI.

Client Component:

1. Client
    - The client component consists on one class DictionaryClient.java that accepts port number and server IP address as command line argument to connect to the client.
    - This is responsible for connecting to server.
    - It has an integrated Graphical User Interface (GUI) that interacts with the user.
    - Through the GUI, there is an option to search, add or delete a word from the dictionary. The request is then sent to the server.
    - Once the response is obtained from server, the result is then displayed in the GUI textArea component.
    - A client can send any number of requests until manual termination of program.

Dictionary file:

The dictionary file is a json file that contains words and their corresponding meanings. The main reason for using json as the format is because of its faster and easy parsing properties. It also has schema support which is easily mapped to an object like notation in java with the

help of external libraries. The syntax is also easy to use and comprehend. The updates to the file are also easy because of the usage of JSONObject which can easily be written into a file. The dictionary file has text in the form of:

{ "word1":["meaning1;meaning2.."], "word2":["meaning1;meaning2…"] ….}

Example: *{"hello":["an expression of greeting"],"laptop":["instrument;device"]}*

The multiple meanings are separated by semicolon and is parsed in the program logic to display appropriately.

Class Diagram

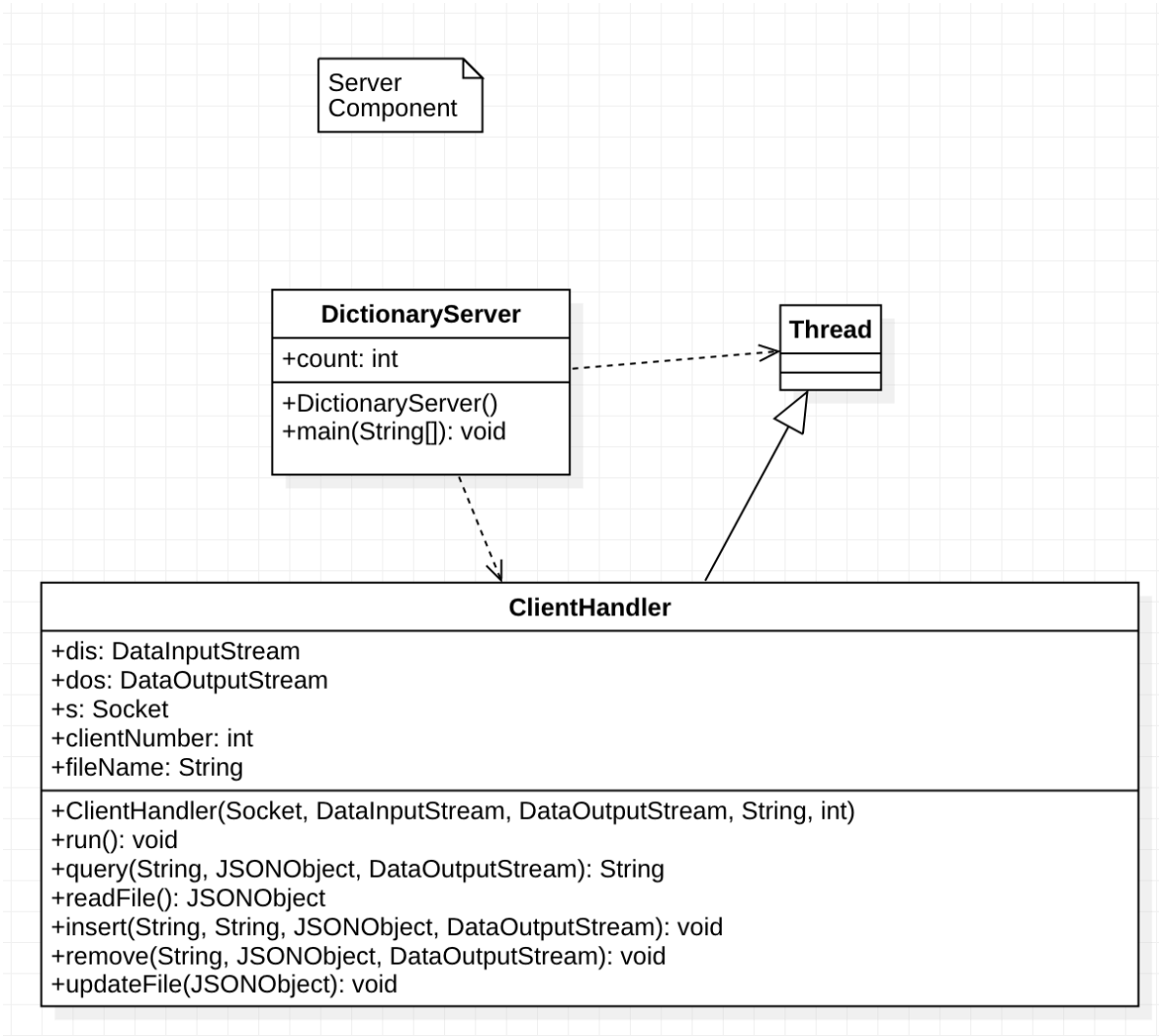

Fig 2: Class diagram representing server component

The above diagram shows us the class diagram of the dictionary server. In the server component, DictionaryServer has the main function which is the initiation point for the server execution. The DictionaryServer class depends on the ClientHandler class and on the java.lang Thread class. This class has a static data member count, that is used to keep a track of the number of clients connected.

The ClientHandler class extends the Thread class. This is to show the multithreading aspect of the server, where for each client a thread would be created and the ClientHandler class would manage the request and response from and to the client. The attributes of this class are the DataInputStream and DataOutputStream objects used for reading and writing into the sockets, the client socket, client number and filename. The member functions include methods for processing client requests.

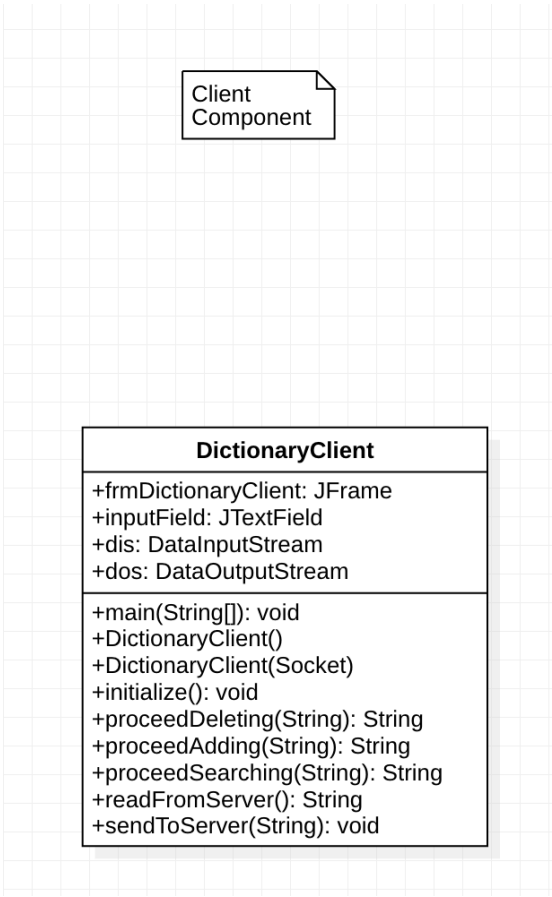

Fig 3: Class diagram representing client component

The client component consists of UI elements as attributes as well as the DataInputStream and DataOutputStream objects to read and write into the socket. As show in the diagram, the member functions are mainly for sending the requests to the server and also for reading the response back from the server.

Sequence Diagram

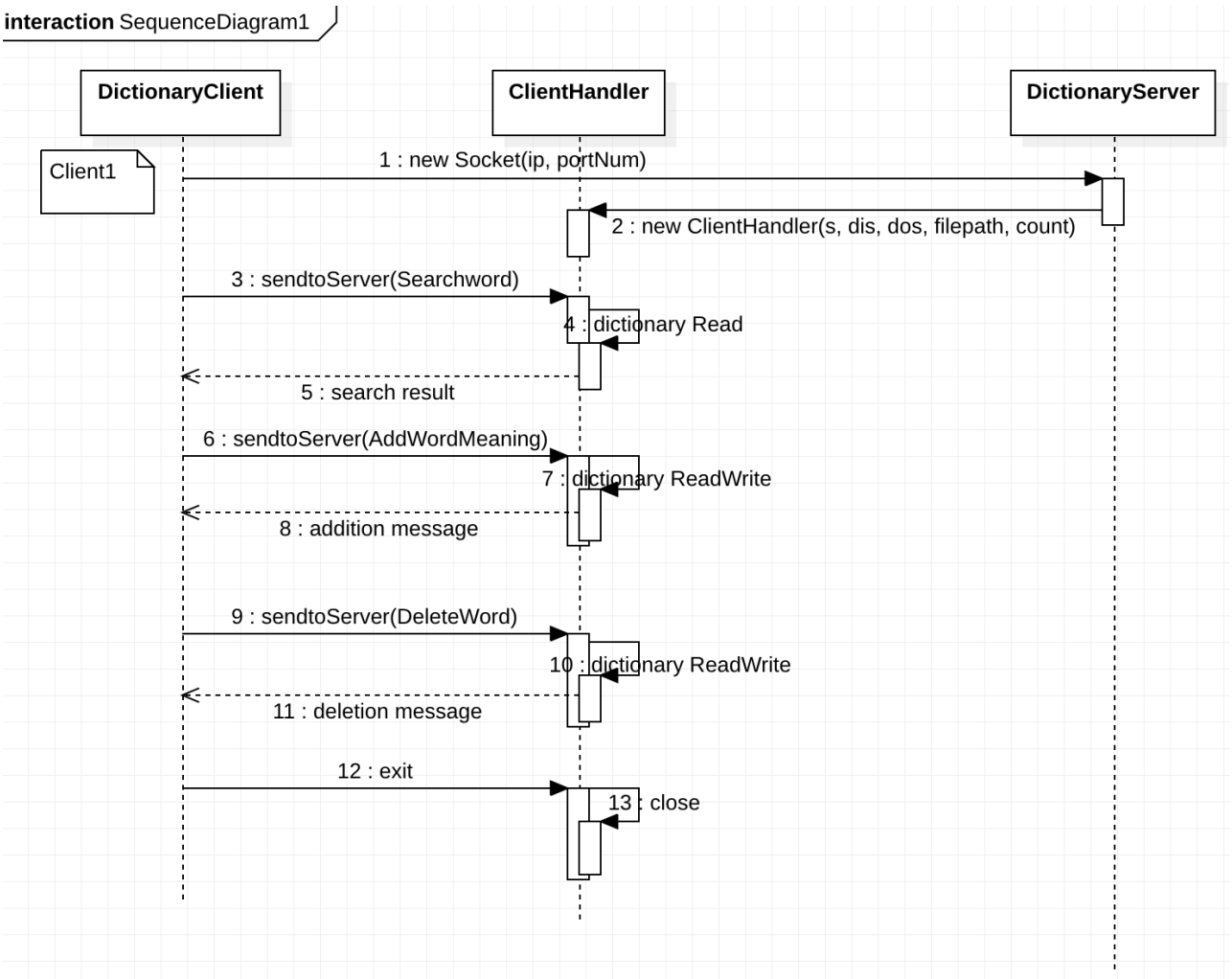

Fig 4: Class diagram representing client component

The sequence diagram shown above indicates the interactions between the client and server components. Messages 1 and 2 are used to establish the connection between the client and server. Messages 3-5 depict the search operation request by the client. The client handler then processes the request by performing a dictionary read and sends the query response back to the client.

Messages 6-8 and messages 9-11 show the addition of word, meaning into dictionary and deleting a word from dictionary respectively.

Message 12 is when the client terminates, following that the ClientHandler will close that particular socket.

Critical Analysis

This project is a working model of a multi-threaded dictionary server. It uses the thread per connection architecture, which efficiently makes use of resources and is able to service multiple clients simultaneously. The server takes port number and dictionary file path as command line arguments and the client takes the port number and the server IP address as command line arguments.

With respect to ensuring consistency of dictionary file, Java's inbuilt support for thread synchronization is used. Therefore, the consistency of dictionary is ensured even with simultaneous updates by different clients. The dictionary file uses the JSON format to store data. This makes update operations very easy with the help of services provided by the org.json external library.

The UI is developed using swing and consists of buttons to choose the desired operation, text field and scrollable text area which can also display long meanings.

There is exception handling in both the server side as well as the client side. The messages are displayed in UI or in the console depending on the case.

Some exceptions that are handled are- *BindException*, *ArrayIndexOutOfBoundsException*, *NumberFormatException, InterruptedIOException, SecurityException, ConnectException, SocketTimeoutException, UnknownHostException, NoRouteToHostException and IOException.*

Excellence Elements

- A vast list of exceptions is considered and are handled by displaying appropriate messages to the user. Example: when server is terminated, message is displayed on the client side.
- The query is processing is done efficiently by the client so as to avoid any overhead by performing unwanted file reads.
- Null input, certain invalid input operations have been on the client side to avoid unnecessary server interaction and hence gives faster response.

Disadvantages:

As there is only one server node it is not scalable to handle a huge number of requests. Even thread creation in Java is expensive. Therefore, for the application to be scalable, more than one server node is recommended which is not implemented in this project.

Conclusion

This project meets all the specification requirements. There is reliable communication between the client and server with the help of TCP protocol. The dictionary server is able to concurrently serve multiple clients through multithreading and all the communication is via sockets. The operations of searching for a meaning given a word, addition of a word and its meaning and deletion of a word are implemented, and the exceptions are handled wherever necessary.