Yifei Yu 945753

Distributed System Assignment 1 Report

Introduction:

In this assignment, we are asked to build a multi-threaded dictionary server. This multi-threaded dictionary server needs to be able to handle multiple clients' connection at the same time. And it needs to allow clients to search the meaning of a word, add a new word, update the meaning of an existing word and remove an existing word.

Architecture Used:

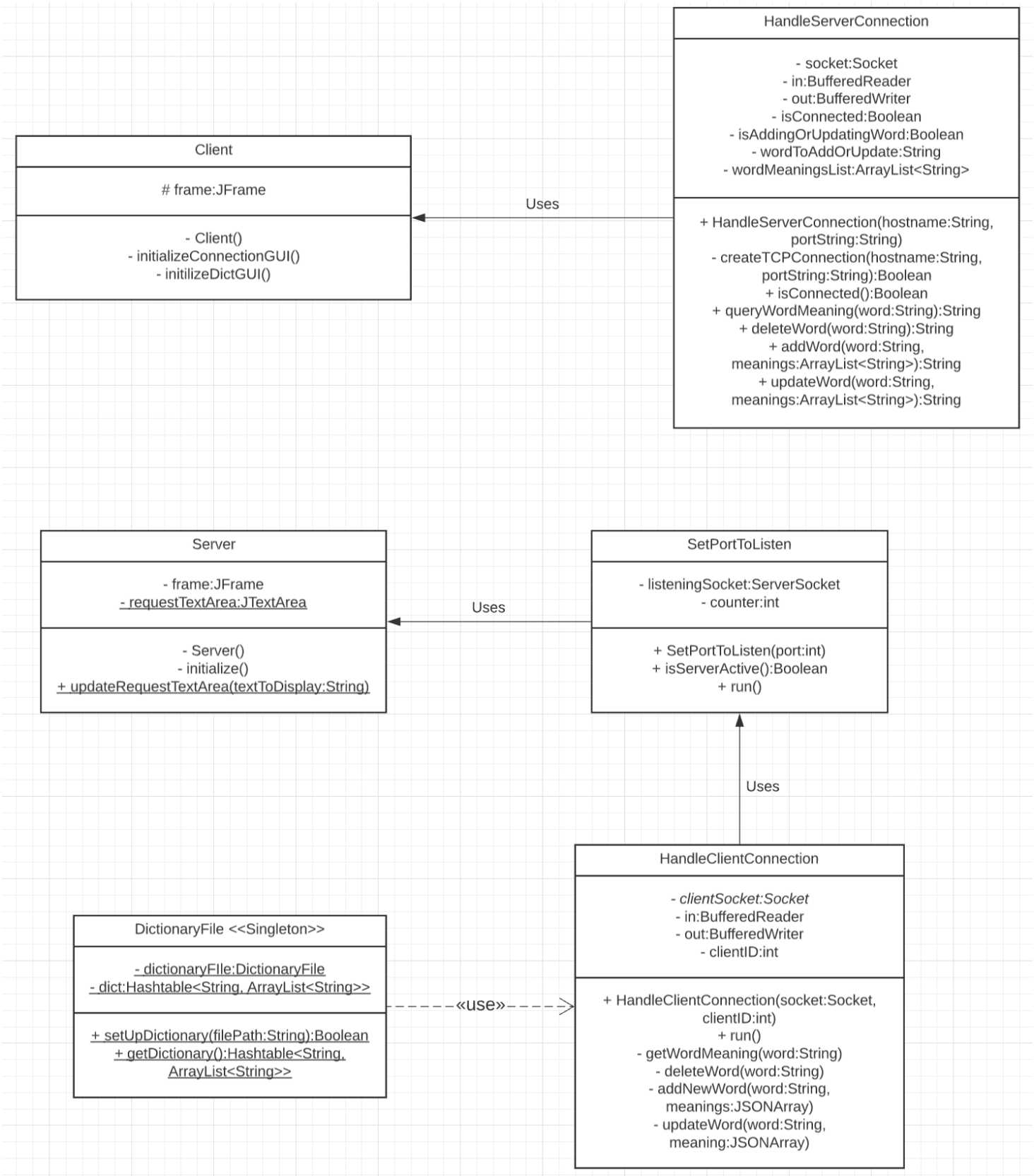
I have implemented the multi-threaded dictionary server using a client-server architecture. This means that multiple clients can be connected to the same multi-threaded dictionary server at the same time.
The thread-per-connection approach is used when designing the system. That is, whenever a new client connects to the server, a persistent connection is established between them. The client can send multiple requests through the same connection. The connection keeps alive even if the client is not sending any requests.

Components of the system:

As you can see from the UML diagram below, the system consists of two parts: the client part and the server part.
The client program consists of two classes, they are the "Client" class and the "HandleServerConnection" class. The "Client" class is for handling and displaying the client GUI. It initializes the "HandleServerConnection" class and stores it as a variable. The "HandleServerConnection" class is responsible for connecting and communicating with the server.
The server program consists of four classes, they are the "Server" class, "SetPortToListen" class, "HandleClientConnection" class and "DictionaryFile" class. The "Server" class is for handling and displaying the server GUI. It initializes the "SetPortToListen" class and stores it as a variable. The "SetPortToListen" class is responsible for setting up a server port, and continuously listening for new connections. The "DictionaryFile" class is a singleton class, it is initialized at the start of the server program, it is used to read a dictionary file that is in JSON format and store it into a Hashtable. The "HandleClientConnection" is created when a new client asks for a connection, it is used to handle the connection and requests coming from the client, and it uses the "DictionaryFIle" class for retrieving the dictionary data.

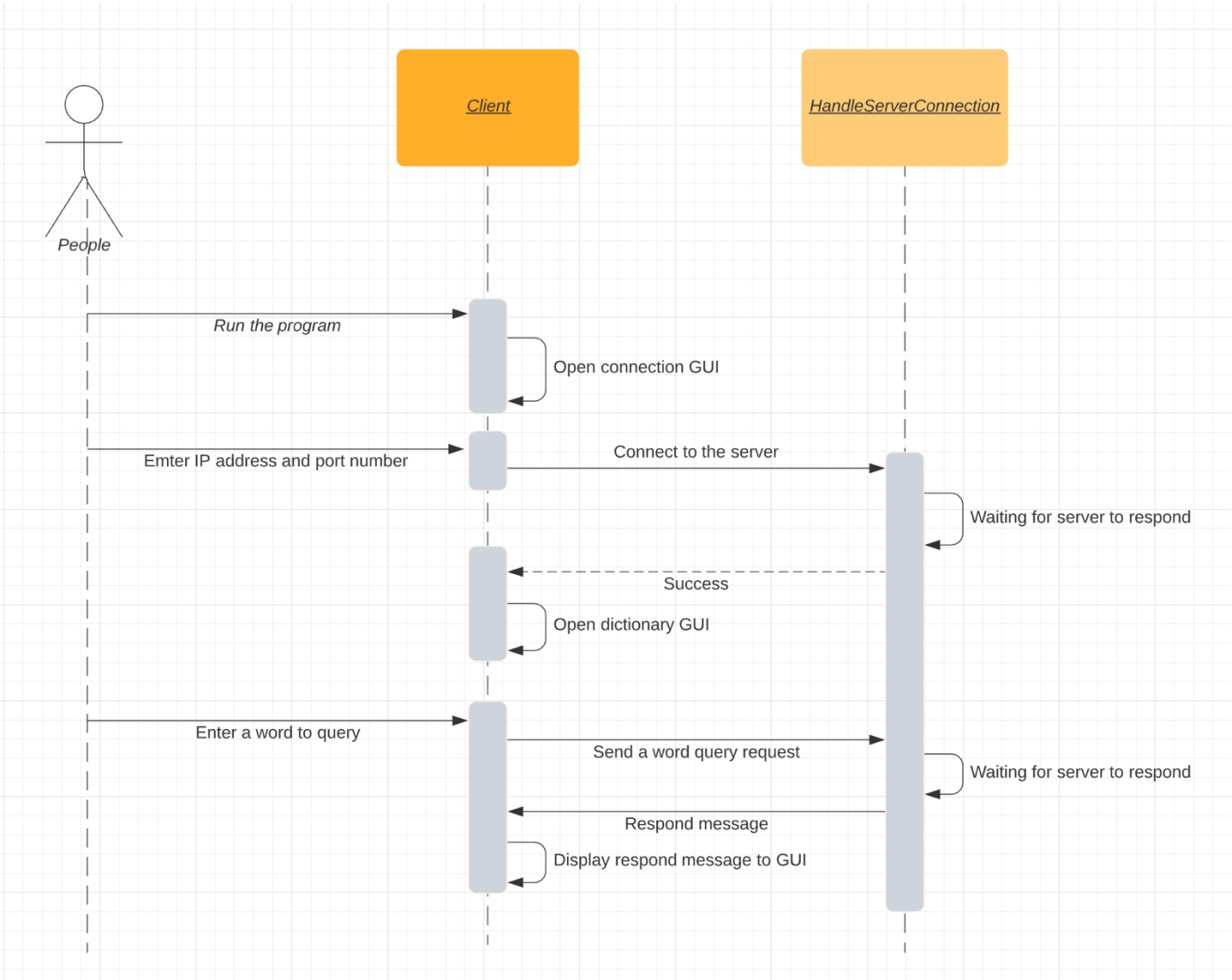
Yifei Yu 945753

UML class diagram:



Interaction/Sequence diagram:

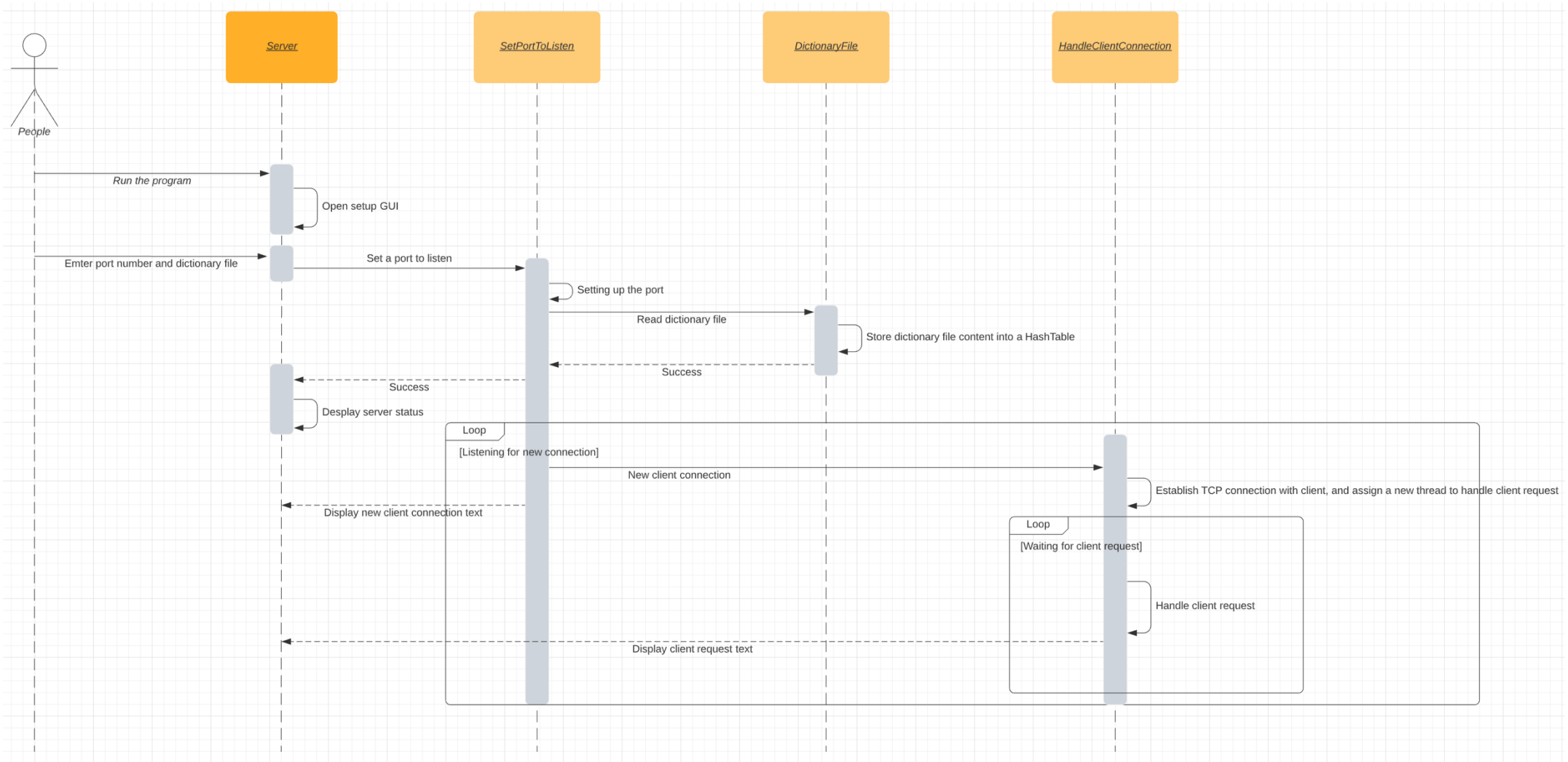
Client:



When running the client program, a GUI will pop up and asking for the dictionary server IP address and the server port. The client will send out a TCP connection to the server. The error message will be prompted if the server IP address or the server port is not correct. Once the TCP connection is established between client and server, an input stream and output stream will be created, and the dictionary GUI will show up. The client will be able to query the meaning of a word, add a new word, remove an existing word as well as update the meaning of an existing word. The client and server communicate by sending JSON files. Error messages will be prompted if any of the issues occur during any of the operations.

Yifei Yu 945753

Server:



When running the server program, a GUI will pop up and asking for the port number to listen to and the dictionary file. The error message will be prompted if the port number is in use or can't find the dictionary file. Once a listening port has been established, a thread will be created to continuously listen for incoming connection requests on that port. The GUI will then display the port number that the server is currently listening to, and there is a console block for showing the id of the connected client. When a client sends a connection request, the server will automatically assign the client a new port for persistent connection, the input and the output stream will also be created. All of the future requests coming from that client will be handled by a new thread. This means that the server can handle multiple clients' connections at the same time. Whenever a client makes a request, it will show up in the server console.

Additional Implementation:

- Client GUI to ask for server IP address and port
- Server GUI to ask for the port to listen and the dictionary file to read, and to display the status of the server as well as the clients' requests

Critical Analysis:

I chose to use TCP protocol to communicate between server and client is that it is reliable. The message is guaranteed to be delivered, this ensures that the dictionary app is reliable, and the user will be able to reliably send request and receive a response to and from the server.

A thread-per-connection architecture is used. The advantage of using the thread-per-connection is that, once the connection is established, the client will be able to send request and receive responses using the same output and input stream. This makes it easy and fast for the server and client to communicate. The downside of using thread-per-connection is that, since each client-server connection will occupy one port and there is a limited number of ports available, this means that the number of clients that can be connected to the server at the same time will be limited to the number of ports available.

Conclusion:

In conclusion, I have used both the sockets as well as the threads to build a multi-threaded dictionary server. It is capable of handling multiple clients' connections at the same time. It can achieve all the functionalities that are required, and the errors are properly handled.