**COMP90015 Assingment 1**

Zongliangh Han

1166050

For this assignment, the task involves creating a dictionary application with key features such as word lookup, adding new words, updating word meanings, and word deletion. To enhance scalability and responsiveness, the application will be split into two distinct parts: a server and a client. Multithreading will be implemented on the server side to make sure it can handle multiple user requests at the same time. Additionally, a straightforward user interface is also implemented to enhance the overall usability of the dictionary application.

**Server Application**

Below is the class design diagram for the server application. We've structured the application into seven classes to increase the cohesion and reduce coupling between components.

A diagram of a computer program

Description automatically generated

1. ServerGUI

The serverGUI class provides an interactive interface for users. It offers buttons to start or stop the server and allows users to input the desired dictionary file and port number. When a user clicks the button to start the server, a new server object will be created.

2. Server

The server class manages the server's core functionality. It handles client connections and stores them as ClientHandler instances in a list. Each clientHandler is able to create a new thread in order provide service for client. When the server is closed, it iterates through all running client sockets, closes them, and terminates the server listening socket.

3. ClientHandler

The clientHandler is created by server when a new connection request arrives. Multithreading is implemented here using a thread-per-connection model. ClientHandler threads analyze incoming requests through the Controller class.

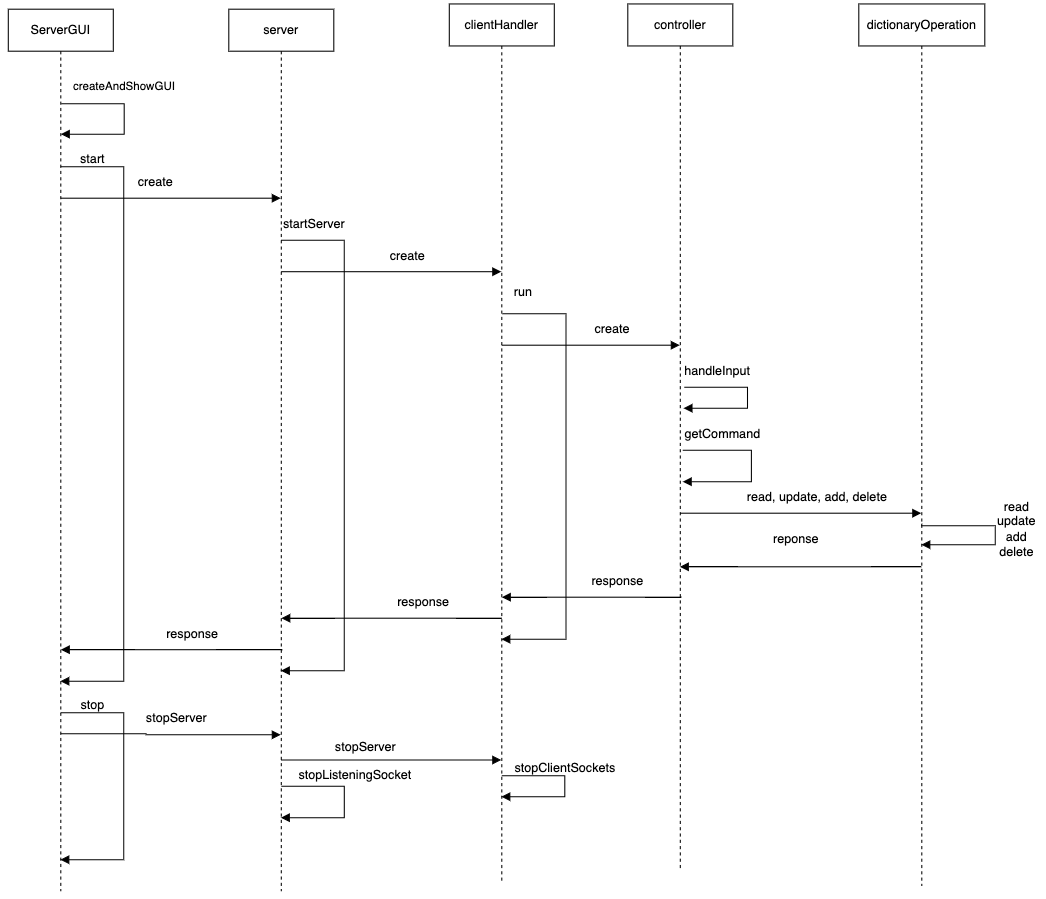
4. Controller

The Controller class is responsible for routing incoming requests to the appropriate functions in the DictionaryOperation class. It acts as an intermediary between the server and the dictionary operations.

5. DictionaryOperation

This class encapsulates the core dictionary operations, such as word lookup, adding new words, updating word meanings, and word deletion. It handles the actual operations and returns the corresponding responses to the Controller.

Below is the interaction diagram of server application, which show the workflow in graphical way.



By structuring the server application in this way, we aim to achieve high cohesion and low coupling, where each class has a well-defined responsibility. By doing this, changes to one class are less likely to affect others. This design also enables concurrent handling of multiple client connections, enhancing the application's responsiveness and scalability.

**Client Application**

Below is the class design diagram for the client application. We've structured the application into nine classes to increase the cohesion and reduce coupling between components.

A diagram of a computer

Description automatically generated

1. ClientGUI

The ClientGUI class provide a user interface for the user. This class create MainPanel, ConnectionPanel and ButtonPanel at start. It works as a place holder for these panels. It will create a client instance based on the input data from connectionPanel.

2. ConnectionPanel

The ConnectionPanel provides a user interface to input the server IP address and server port number. these data will be returned to ClientGUI for initialize client instance.

3. Client

The Client class manages the core functionalities of client application. It is responsible for establishing the connection with server and sending the request with information collected from user interface.

4. ButtonPanel

The ButtonPanel is a place holder for read, add, update, delete button, as well as refresh button. It enables user to switch from deferent panels which contain different functionalities. In addition, the refresh button will try to create a new client instance and establish a new connection channel with server when the current connection is lost.

5. MainPanel

The MainPanel is a place holder for addPanel, readPanel, deletePanel and updatePanel. It uses CardLayout to organize these panels.

6. AddPanel, UpdatePanel, readPanel, deletePanel

These four classes work as user interfaces for distinct functionalities. They enable the utilization of a client instance to transmit requests based on user input data. Dividing these functions into separate panels enhances the usability of the client application, making it more user-friendly and intuitive.

Below is the interaction diagram of client application, which show the workflow in graphical way.

A diagram of a software

Description automatically generated

By organizing the client application in this manner, we achieve a structured arrangement of UI components, each residing in a separate class with well-defined responsibilities and functionalities. This design approach serves to minimize coupling between components while enhancing their cohesion. Moreover, the use of distinct function panels contributes to a user-friendly interface, providing users with a more intuitive and organized experience.

**Database**

In this assginment, I use JSON file as mock database. It follow the format below:

[

{

“word”: “run”,

“meaning”: ["v. to move quickly on foot", "n. a continuous period of operation"]

},

{

“word”: “watch”,

“meaning”: [ "v. to observe", "n. a timekeeping device", "n. a period of time spent observing”]

}

]

The Word class has been designed as a model to facilitate the extraction of data from JSON files by using the jackson library. It also enables the modification of word meanings. Furthermore, data validation has been implemented within the user interface. This validation mechanism ensures that only correctly formatted data is allowed to proceed and be sent as requests, thereby upholding data integrity and preventing incorrect data from entering the system.

**Error Handling**

In the application, there are several error could be identifyied, including wrong IP/port address, lost connection with server, adding existing word into database, updating non-existing word.

The system has been equipped to effectively handle these errors, responding with appropriate error messages displayed in the user interface window. This comprehensive error-handling mechanism ensures that users are informed of any issues and can interact with the application more effectively.

**Advanced Implementation**

The user interface for the server application has been successfully implemented, giving users the ability to initiate or terminate the server through dedicated buttons. Additionally, users have the flexibility to customize the port address and file location by directly inputting their desired values into corresponding text fields.

**Analysis & Conclusion**

Both the client and server applications have been carefully designed to adhere to the principles of low coupling and high cohesion. Each class within the applications possesses well-defined responsibilities and functionalities. However, it's worth noting that certain static methods, such as “sendRequest()” and “refresh()”, have been employed. While these methods are functional, the design has been refined to minimize the need for static methods.

Additionally, the introduction of multithreading in the connection-per-thread model is a noteworthy enhancement. Changing to the request-per-thread approach will significantly improve parallelism and diminishes user waiting times. Furthermore, implementing of a worker pool during the server's initialization phase will serves to further reduce user wait times, as it eliminates the delay typically associated with thread creation. This will significantly increase the application’s usability and scalebility.

Furthermore, the application can be extended to connect with a real database instead of relying on the current JSON file. This enhancement not only allows users to store a large volume of data but also ensures quicker access times. Importantly, the application's design facilitates this modification seamlessly, as adjustments can be made within the “DictionaryOperationImp” class without disrupting the functionality of other classes.

In conclusion, the client and server application were implemented with the design principle of low coupling and high cohesion. The user interface is also developed for both server and client application. The server manages client connections using multithreading by thread per connection model. Client can fulfill CRUD operations in different panels by sending request to the server. Identified error message will also be show on the board. The usability and scalability of the system can be enhanced by changing the multithreading model and connecting with a real database.