# COMP90015 Distributed Systems

# Assignment 1 Multi-threaded Dictionary

# Ziming Wang (1180051)

Contents

[1.Discussion of the problem context 1](#_Toc144056047)

[2.Descriptions of the components of system 2](#_Toc144056048)

[3.Overall class design and interaction 2](#_Toc144056049)

[a. Java Version and Dependency 2](#_Toc144056050)

[b. Transport Layer Protocol 2](#_Toc144056051)

[c. Message exchange protocol 2](#_Toc144056052)

[d. Dictionary and its CRUD workflow 3](#_Toc144056053)

## 

## 1.Discussion of the problem context

The assignment requires a java multi-threaded dictionary that follows a client-server architecture. The lowest communication protocol should be at transport layer, e.g., TCP/UDP. The server should have multi-threaded capabilities in order to handle multiple concurrent requests. The dictionary should support the word CRUD operations with reasonable input validations. It loads initial data from a file and modifies the content dynamically while executing. Additionally, an interactive GUI is required for client. Finally, all errors should be reported accordingly and the integrity of the client-server should be preserved.

## 2.Descriptions of the components of system

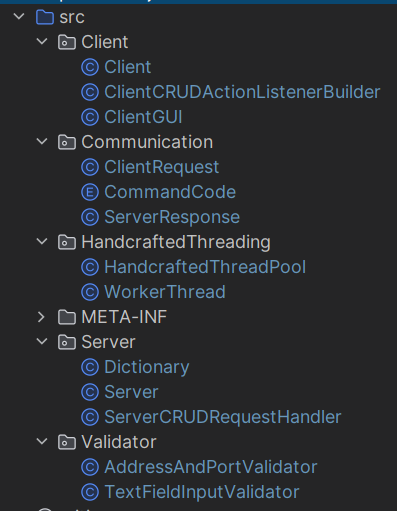
Figure 1 demonstrates the components of the systems.

Figure -code directory

“/Client” and “/Server” package contains corresponding core logic code for client and server.

“/Communication” package contains the communication protocols for client and server to communicate. For example, the command code sent from client to server to perform a CRUD operation. This will be discussed thoroughly in later sections.

“/Validator” package contains the input validator for both server and client to use. For example, both server and client will use the “AddressAndPortValidator.java” to validate the port from command line arguments. They will use “TextFieldInputValidator.java” to validate the user input data such as word and meanings.

“/HandcraftedThreading” package contains my handcrafted worker pool multi-threading logics. This will be discussed in later sections.

## 3.Overall class design and interaction

### a. Java Version and Dependency

My java version is corretto-17 (Amazon Corretto Distribution).

I have an external dependency from org.json to handle json files. The dependency can be downloaded at <https://jar-download.com/artifacts/org.json>.

### b. Transport Layer Protocol

I chose to use TCP socket over UDP as transport layer protocol driven by the need for message reliability. For my application, where reliable message exchange is crucial, the use of TCP ensures that critical messages are handled accurately and efficiently.

### c. Message exchange protocol

The server needs to handle CRUD (create, retrieve/read, update, delete) operations. I have set up an enumeration for command code for client to make a request as demonstrated in Figure 2.

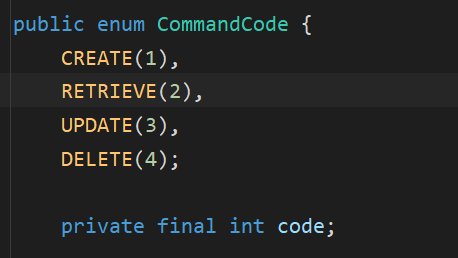


Figure -ENUM command code (/Communication/CommandCode.java)

With the command code as part of the field, the client can send a ClientRequest with fields including “commandCode”, “word”, and “meaning” through socket as demonstrated in Figure 3. Java will automatically serialize and deserialize the object. All ClientRequest must have “commandCode” field and “word” field. Additionally, CREATE and UPDATE request must have “meaning” field. Otherwise, this field should be null.

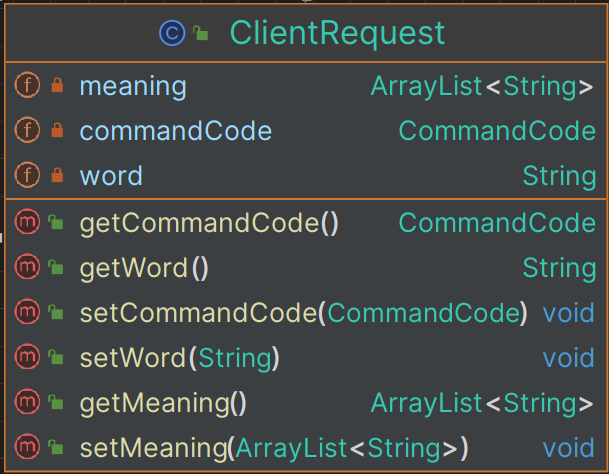


Figure -ClientRequest (/Communication/ClientRequest.java)

Upon receiving the ClientRequest, the server will response with a ServerResponse that contains three fields, “successful”, “word”, and “meanings” as demonstrated in Figure 4. The “successful” field indicates if the CRUD operation is a success or failure. For example, if the client wants to query a word that doesn’t exist in the dictionary, or wants to create a word that has already exist, this field will be set to false. Additionally, if the server receives a RETRIEVE request and the operation is performed correctly on the dictionary, then the server will also response with the “meanings” field set as the meaning of the word.

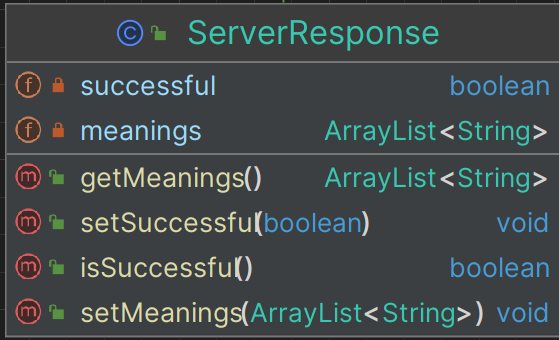


Figure -ServerResponse (/Communication/ServerResponse.java)

### d. Dictionary and its CRUD workflow

I use json to build the dictionary and perform operations on it with the external dependency from org.json. Figure 5 presents the fields and methods of my dictionary. Specifically, I maintain an in-memory JSONObject data structure which is loaded from a json file on disk. Whenever there is CREATE/UPDATE/DELETE action performed, the in-memory JSONObject will be modified accordingly, and then the json file on disk will be updated.

All CRUD methods on the dictionary are synchronized, in order to suit the design of concurrency.

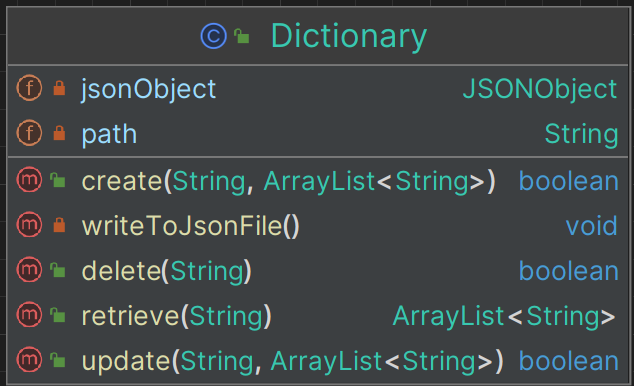


Figure -Dictionary (/Server/Dictionary.java)

Figure 6 demonstrates the sequence diagram of CREATE request workflow for server and dictionary. The “/Server/ServerCRUDRequestHandler.java” receives the request and creates a “/Server/ServerResponse.java” object. Then, it invokes the “handleCreateRequest” method, and performs input validation on word and meanings by invoking the “/Validator/TexFieldInputValidtor.java”. If all validations are passed, the “create” method of the “/Server/Dictionary.java” will be invoked, and a new entry will be created in the in-memory JSONObject, followed by an update to the json file stored on disk.

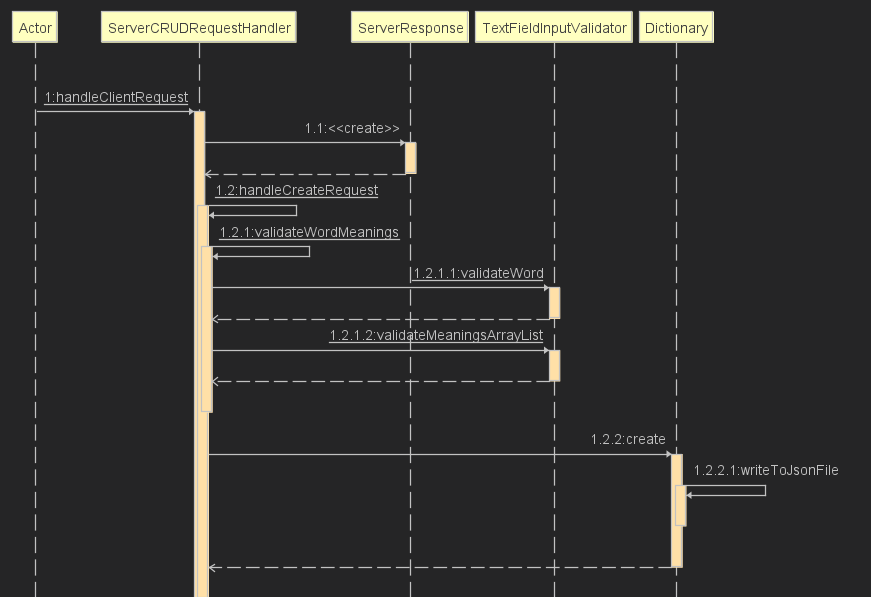


Figure -Sequence Diagram, server handles client create requests

## 4. Highlights and Creativities

### a. Own implementation of thread pool for creativity marks

I used my own implementation of thread pool instead of java built-in one. The relevant files can be found in directory “/HandcraftedThreading/HandcraftedThreadPool.java” as shown in Figure 7. The structure follows a worker pool architecture. When initializing the thread pool, a “poolSize” will be set up that indicates the number of work threads. A task queue is set up to store all pending tasks. When the client sends a request to the server, the server will set up a new runnable task to the taskQueue to let it wait for the execution. While the worker threads (WorkerThread.java) will retrieve task from this task queue to run them.

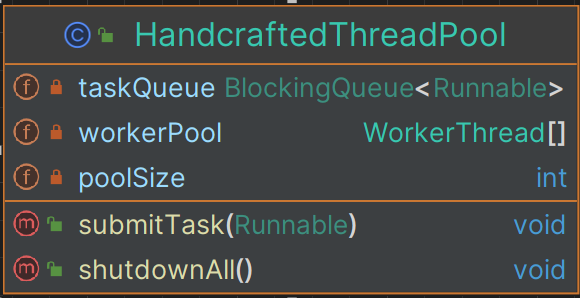


Figure -(/HandcraftedThreading/HandcraftedThreadPool.java)

**Critical discussion:** Using a worker pool architecture for thread management has both advantages and disadvantages. It reduces overhead of creating and destroying threads. It also makes computational resource more controllable and measurable, hence increases scalabilities. Besides, it could prevent the server from overflowing in a high concurrency situation. However, if the pool size is set to be too small compared with server’s capability, then during the time of high traffic, the server’s resource may not be fully utilized and the response time may be long. In other word, it lacks of dynamic scaling if using a fixed pool size. To improve this, a scalable pool size adjustment approach can be used. In addition, the management overhead of using worker pool architecture in small-scale application might outweigh the benefits, due the small number of requests.

### b. Security Highlights

Both client and server should have comprehensive security measures. In general, the ideal use case is that a user uses the client GUI to send a request to the server. Therefore, the client should undoubtfully have comprehensive security measures such as input validations. However, a skillful malicious user could exploit the data package send from the client and write scripts to send request to the server directly. As a result, the security measures in the client GUI can be bypassed. Therefore, it is equally important to have same level of security on both the client and server. In my implementation, both client and server use the validators located at “/Validators”. For example, for the CREATE request, the client GUI will validate the user’s input and deny any invalid format using “TextFieldInputValidator.java”. The server will also use it to perform the validation on the ClientRequest payload to double check if the word/meaning is legit.