Project2 Executive Summary

Assignment Overview

The major purpose of project 2 is to understand how to enable client and server to communicate using Remote Procedure Calls(RPC) instead of using socket and I/O streams to explicitly pass data and information. Through the implementation process of utilizing Java Remote Method Invocation(RMI), we were able to step through the remote invocation paradigms by seeing how RMI extends the concept of an object reference to the global distributed environment, and allows the use of object references as parameters in remote invocations. Furthermore, we explored the difference between socket programming and RPC, knowing the enhancements that RPC can bring to distributed environments. RPC supports more concise and powerful communication between server and client, and can easily support multi-threaded servers with mutual exclusion handled.

In addition, practices contain:

1. **Design and programming:**
- Java RMI usage
- Java multi-thread programming with handling mutual exclusion
- Java object-oriented design
- Code refactoring
2. **(Still) Learning to use new tools:**
- Docker(learn to use a container, pack the application to fit different Operating Systems aligns with the idea of building distributed systems)
- Shell scripts

Technical Impression

Features:

1. **Put** operation: for users to put a key-value pair into the store. Putting in an already-in-the-store key with a new value will override the old value.
2. **Get** operation: for users to get the value by giving a specific key. Getting a key that is not in the store will prompt a warning log message to the user.
3. **Delete** operation: for users to delete a key-value pair into the store by giving a specific key. Deleting a key that is not in the store will prompt a warning log message to the user.
4. **Multi-threaded Server:** The project supports a multi-threaded server and it can handle multiple requests from different clients.
5. **Log messages**: The program includes human-readable log messages for users to interact with correct commands input.

Assumptions & Design Choices:

1. OOD Design structure clarifications

The **client** package includes:

- Client App/Controller with the main method: *ClientController*
- Logs print out in human-readable format with timestamps of current system time: *ClientLogger*

The **server** package includes:

- KeyValue store and operations with related files: interface *KeyValue*, class *KeyValueStore*
- Server App/Controller with the main method: *ServerController*
- Logs print out in human-readable format with timestamps of current system time: *ServerLogger*

2. Protocol Design and Assumptions

Currently for handling response/request parsing, since there is no specific requirement for the project, I used a super rough design - performing regular expressions with strings to deal with requests by splitting them with space. As for responses, I used ";" as a separator to handle code and response repsectively. This was highly based on the assumption that we only include a limited type of requests, code and reponses.

3. Implementing details(more can be found in comments and documentation in code)

- For the put method, putting in an already-in-the-store key with a new value will override the old value.
- For this multi-threaded project 2, prepopulated data sets operations will be performed at the time that any client starts.

Enhancements for future works:

1. Scalability and Modularity consideration for Protocol Design. Instead of handling strings and performing regular expressions, we can create classes for request and response to handle them specifically. Thus in future implementations if we need to handle a larger number/type of requests and responses, we have more flexibility on customizing the design.

2. Scalability consideration for multithreaded servers and ensuring it can always be a thread-safe program especially when the situations are not as simple as current. In the future, we may be required to handle synchronization based on specific conditions, so setting locks explicitly may be necessary.