The assignment is based on implementation of the Memcache system, a distributed memory management system. The Memcache system reduces database overhead by loading and retrieving key value pairs in an efficient way. The assignment is a lite version implementation of the memcache server, storing and retrieving key-value pairs from concurrently connected clients.

Details on how to run can be found in README.txt

**Design Details -**

There are two parts to the system, the client and the server. While a server is listening on a port (5001), it can accept multiple client connections and serve multiple clients concurrently. The communication is achieved through socket programming, where a socket acts like an endpoint. Concurrency is achieved through multithreading. Once the server socket is created and the server is listening on the desired port, as a client connects to the port, the server accepts the connection, thus creating a new thread which is executed concurrently.

**Design Decisions**

The client can execute either SET, GET or EXIT command. Any other input will output INVALID COMMAND.

**SET**

* set \r\n value \r\n
* Checks for length of the command string, if <4 output INVALID COMMAND
* If the length entered value is greater than value-size-bytes output INVALID COMMAND
* The value length is less than value-size-bytes outputs NOT-STORED, else outputs STORED
* If a key already exists, the SET command will override the previous value

**GET**

* get key\r\n
* Checks for length of the command string, if <2 output INVALID COMMAND
* If the key exists, returns VALUE <key> <bytes> \r\n
* If key does not exist, outputs INVALID key

ConcurrentHashMap<Object,Object> is used to store key-value pairs. In order to persist data, the HashMap object is serialized and written to a keyValueStore.txt file.

**Performance Evaluation**

* Tested the server for 500 connections. It runs at least 500 connections. Test Cases can be found in keyvaluestore/src/main/java/client/.
* Did not encounter a connection refused on Server side for at least 500 client requests. Also, tested concurrently with one loop creating 500 clients and other creating 200 clients (Tests.java and Test2.java respectively).

**Errors and Future Scope**

Currently, the server throws a java.io.EOFException on EXIT, which is not handled. However, this exception does not hinder the performance and is a scope for further improvement.

Future scope includes adding logging using log4j library in Java.