Sherly Hartono

hartono.s@northeastern.edu

“Assignment overview” (1 paragraph, up to about 250 words)

The purpose of the assignment is to develop a single-threaded key-value store server program and a corresponding client program that can communicate with the server using either UDP or TCP protocols. The client and server must be able to perform three basic operations: PUT (key, value), GET (key), and DELETE (key). The client program must take the server's hostname or IP address and port number as command line arguments and must be robust to server failure with a timeout mechanism. The client must also be able to handle malformed or unrequested packets. The server must listen for incoming datagram packets on a specified port number and display received requests and responses in a human-readable format. The server must also be robust to malformed datagram packets.

The client must pre-populate the key-value store with data. In my case the client will read commands from a txt file and perform at least five of each operation. The client and server will be evaluated on how well they communicate with each other and their compliance with the given requirements.

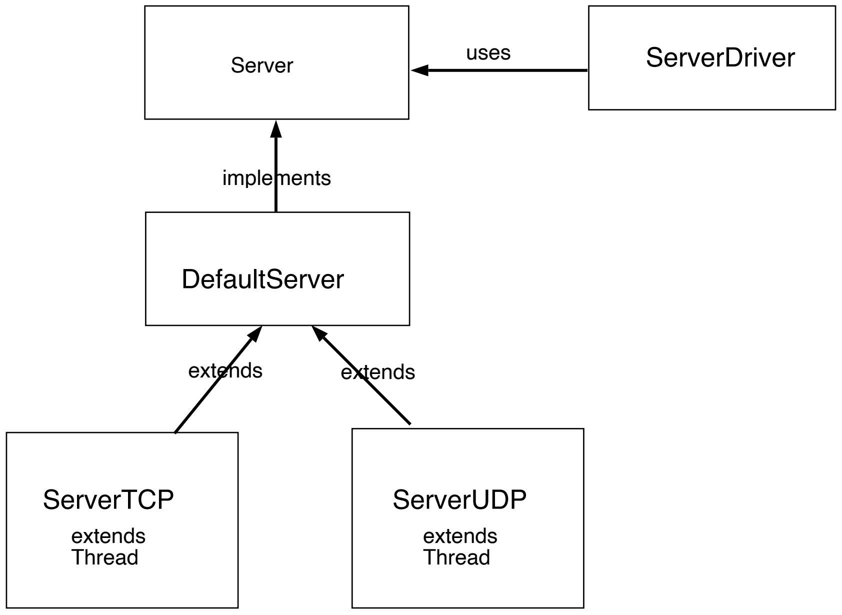
“technical impression” describing your experiences while carrying out the assignment (1–2 paragraphs, about 200–500 words)

1. TCP and UDP as separate classes
In the beginning I wanted to make completely separate classes for TCP and UDP clients and server but I realize I will be rewriting a lot of code and it will be difficult to keep track at which point I want to delete the code for the TCP connection and replace it with UDP. So I decided to create an interface called Server and Client which is implemented by a concrete class called ServerDefault and ClientDefault class. These interfaces have common methods that will be used by TCP and UDP Client and server.
In the Server interface I have getDate(), validateClientInput(), startThread() that will be used by both the TCP and UDP server.
In the Client interface I have startClient(), validateCommand(), getDate(), generateUniqueId(), printResponse(), validateResponse(), convertJsonToMap(). As you can see we have many common methods so making a common interface keeps our code DRY.

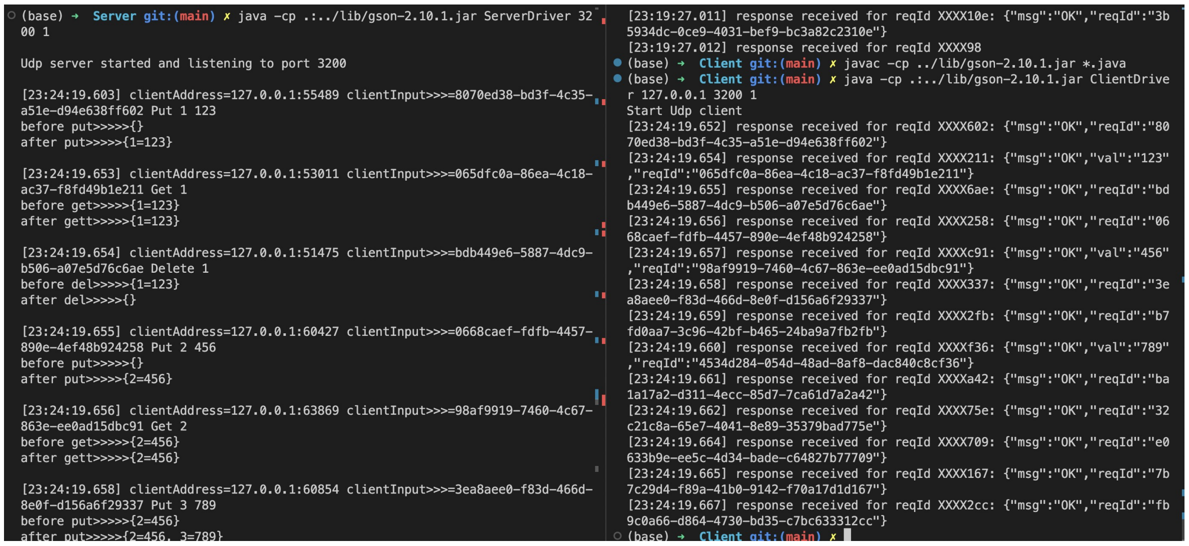
2. Command class
To further keep my TCP and UDP server code simpler, I also created another class called Command. Which will take care of all the logic given an input of command passed by our client. In its go() method, it will pass the arguments of the commands to the respective command and create a response string. It will then create a Gson object out of the string.

3. return Get value and the status response together
I used Gson libraries created by Google as a way to imitate JSON since I don’t want to parse the string everytime the client receives the response which is bound to give a lot of error. This is because I want to pass the following as a response back to the client: the request id from the client, the status result, and whichever value that is returned by the method. If the order is wrong, there will be many dropped packets. So I figured making it as JSON will reduce this risk of wrong parsing order.

4. Have to rewrite several methods such as: validateCommand(), getDate(), isWordNumeric() in both server and client.
I have to repeat these three methods in client and server code. For now I don’t have a way around this since we want the client to be completely detached from the server. I am thinking maybe in RMI or RMC clients can just call validateCommand of the server and basically have them do the same thing once.



Screenshots of results



```
(base) ➜  Server git:(main) ✗ java -cp .:../lib/gson-2.10.1.jar ServerDriver 32
00 1

Udp server started and listening to port 3200

[23:24:19.603] clientAddress=127.0.0.1:55489 clientInput>>>=8070ed38-bd3f-4c35-
a51e-d94e638ff602 Put 1 123
before put>>>>>{}
after put>>>>>{1=123}

[23:24:19.653] clientAddress=127.0.0.1:53011 clientInput>>>=065dfc0a-86ea-4c18-
ac37-f8fd49b1e211 Get 1
before get>>>>>{1=123}
after gett>>>>>{1=123}

[23:24:19.654] clientAddress=127.0.0.1:51475 clientInput>>>=bdb449e6-5887-4dc9-
b506-a07e5d76c6ae Delete 1
before del>>>>>{1=123}
after del>>>>>{}

[23:24:19.655] clientAddress=127.0.0.1:60427 clientInput>>>=0668caef-fdfb-4457-
890e-4ef48b924258 Put 2 456
before put>>>>>{}
after put>>>>>{2=456}

[23:24:19.656] clientAddress=127.0.0.1:63869 clientInput>>>=98af9919-7460-4c67-
863e-ee0ad15dbc91 Get 2
before get>>>>>{2=456}
after gett>>>>>{2=456}

[23:24:19.658] clientAddress=127.0.0.1:60854 clientInput>>>=3ea8aee0-f83d-466d-
8e0f-d156a6f29337 Put 3 789
before put>>>>>{2=456}
after put>>>>>{2=456, 3=789}
```

```
[23:19:27.011] response received for reqId XXXX10e: {"msg":"OK","reqId":"3b
5934dc-0ce9-4031-bef9-bc3a82c2310e"}
[23:19:27.012] response received for reqId XXXX98
(base) ➜  Client git:(main) ✗ javac -cp ../lib/gson-2.10.1.jar *.java
(base) ➜  Client git:(main) ✗ java -cp .:../lib/gson-2.10.1.jar ClientDrive
r 127.0.0.1 3200 1
Start Udp client
[23:24:19.652] response received for reqId XXXX602: {"msg":"OK","reqId":"80
70ed38-bd3f-4c35-a51e-d94e638ff602"}
[23:24:19.654] response received for reqId XXXX211: {"msg":"OK","val":"123"
,"reqId":"065dfc0a-86ea-4c18-ac37-f8fd49b1e211"}
[23:24:19.655] response received for reqId XXXX6ae: {"msg":"OK","reqId":"bd
b449e6-5887-4dc9-b506-a07e5d76c6ae"}
[23:24:19.656] response received for reqId XXXX258: {"msg":"OK","reqId":"06
68caef-fdfb-4457-890e-4ef48b924258"}
[23:24:19.657] response received for reqId XXXXc91: {"msg":"OK","val":"456"
,"reqId":"98af9919-7460-4c67-863e-ee0ad15dbc91"}
[23:24:19.658] response received for reqId XXXX337: {"msg":"OK","reqId":"3e
a8aee0-f83d-466d-8e0f-d156a6f29337"}
[23:24:19.659] response received for reqId XXXX2fb: {"msg":"OK","reqId":"b7
fd0aa7-3c96-42bf-b465-24ba9a7fb2fb"}
[23:24:19.660] response received for reqId XXXXf36: {"msg":"OK","val":"789"
,"reqId":"4534d284-054d-48ad-8af8-dac840c8cf36"}
[23:24:19.661] response received for reqId XXXXa42: {"msg":"OK","reqId":"ba
1a17a2-d311-4ecc-85d7-7ca61d7a2a42"}
[23:24:19.662] response received for reqId XXXX75e: {"msg":"OK","reqId":"32
c21c8a-65e7-4041-8e89-35379bad775e"}
[23:24:19.664] response received for reqId XXXX709: {"msg":"OK","reqId":"e0
633b9e-ee5c-4d34-bade-c64827b77709"}
[23:24:19.665] response received for reqId XXXX167: {"msg":"OK","reqId":"7b
7c29d4-f89a-41b0-9142-f70a17d1d167"}
[23:24:19.667] response received for reqId XXXX2cc: {"msg":"OK","reqId":"fb
9c0a66-d864-4730-bd35-c7bc633312cc"}
(base) ➜  Client git:(main) ✗
```