Gregory Friedman
5511-4893
friedmangreg37@ufl.edu

To compile the code in Linux:

javac *.java

To run the code in Linux, for the server:

java server [port_number]

For example:

java server 4893

To run the code in Linux, for the client:

java client [serverURL] [port_number]

For example:

java client localhost 4893

Structure of server program:

The program starts by checking to see if an argument is supplied in the command line for the port number. If not such argument is supplied an exception will be thrown. Otherwise, it creates a socket at this port number. It then enters an infinite loop, waiting for a client to connect to it. Once it's connected, it prints a message stating this and then sends "Hello!" to the client, creating an input and output stream connected to the client. Now that it's connected, it enters another infinite loop, which runs until the client closes the connection. In this loop, it reads input from the client and then decides what to send back to the client – either the result of a valid operation or an error code. If the client sends "bye" the server will exit the inner infinite loop and then wait for connection from another client. If the client sends "terminate" the server will exit the entire program.

Structure of client program:

The program starts by checking to see if arguments are supplied in the command line for the url and the port number. If not, then an exception is thrown. Otherwise, it creates a socket with the first argument as its url and the second argument as its port number, to connect to a server. It creates an input stream to read input from the user, along with an input an output stream to read from and send to the server. Once it is connected to the server, it will read the input from the server ("Hello!") and print it. Then it will enter an infinite loop until the user wants to quit. In the loop, it reads input from the user then sends this input as is to the server. It reads the response from the server and based on this response decides what to print. If the response is a nonnegative number the client will just print this result. If it is negative, it is an error code, so the client will print the corresponding error message. If the error code is -5, "bye" or "terminate" was the input from the user, so the client breaks out of the infinite loop, closes the socket, and ends execution. Otherwise, it loops through again.

Results:

Server:

```
^Csun01:14% java server 4893
get connection from 70.171.42.129
```

```
get: bye, return: -5
get connection from 128.227.205.198
get: terminate, return: -5
sun01:15% java server 4893
get connection from 128.227.205.198
get: add 34 54, return: 88
get: subtract 98 3 45, return: 50
get: multiply 3 56 2 14, return: 4704
get: add 4, return: -2
get: subtract 23 5 4 3 2 1, return: -3
get: multiply, return: -2
get: lets test to see what will happen, return: -1
get: add 45 d 34 12, return: -4
get: subtract 96 4 34 2 d, return: -3
get: bye, return: -5
get connection from 128.227.205.198
get: terminate, return: -5
```

Client:

```
sun01:3% java client sand.cise.ufl.edu 4893
receive: Hello!
add 34 54
receive: 88
subtract 98 3 45
receive: 50
multiply 3 56 2 14
receive: 4704
add 4
receive: number of inputs is less than two
subtract 23 5 4 3 2 1
receive: number of inputs is more than four
multiply
receive: number of inputs is less than two
lets test to see what will happen
receive: incorrect operation command
add 45 d 34 12
receive: one or more of the inputs contain(s) non-number(s)
subtract 96 4 34 2 d
receive: number of inputs is more than four
bye
receive: exit
sun01:4% java client sand.cise.ufl.edu 4893
receive: Hello!
terminate
receive: exit
```

I think the program works as it should – it gives the correct results for each of the operations and all of the error codes work properly. It also gives the errors in the correct order of priority. It correctly closes the client program when the user enters either “bye” or “terminate” and closes the server program when the user enters “terminate.” There might be a problem in the handling of terminate – I’m not positive that I did this correctly. It closes the calling client and the server,

and then the program exits. Any other clients that may be trying to connect would then throw an exception because the server socket is closed, so I catch the exception but then do nothing, allowing the client's program to exit. Then there are no exceptions in the terminal, and I don't think any processes are left running – I try running "killall java" and it says that no matching processes are found. So I think I am doing this correctly and allowing the program to exit gracefully, but I'm not positive.