

Robert Henderson

- Source code for your client
- Source code for your server
- Description of the protocol as implemented

The protocol is quite simple. I do all the validation of the responses server side. This is important because I wanted to make the client as simple and lightweight as possible. If someone else comes along and wants to write their own client to talk to my server, then by all means. They can make it all fancy and add a nice GUI as much as they desire. The important part here is that the server handles the requests properly performing all the validation. In a typical protocol, such as ftp or a database connection the users can quit/terminate the protocol at any time. I made sure the user would be able to do that here.

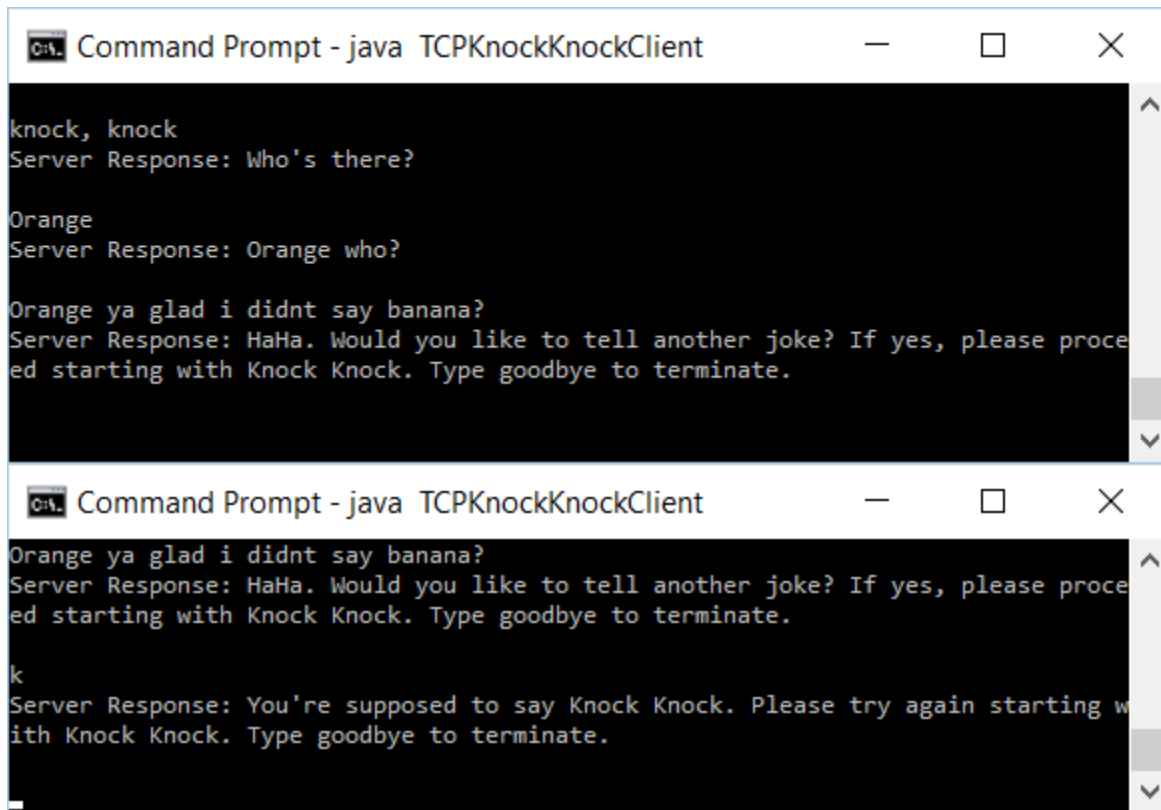
The users should be able to properly use the protocol from the client perspective. If they are unable to, I have the protocol/connection resetting itself and it tells the user how to properly use it.

Say for example, they don't know to start with "Knock Knock". I have a message telling them the proper usage.

```
Command Prompt - java TCPKnockKnockClient
C:\Users\Robert\Desktop\infs>java TCPKnockKnockClient
The client is up and running
k
Server Response: You're supposed to say Knock Knock. Please try again starting with Knock Knock. Type goodbye to terminate.
```

I also made the input case insensitive and even allowed for the user to use a comma – knock, knock

```
Command Prompt - java TCPKnockKnockServer
C:\Users\Robert\Desktop\infs>java TCPKnockKnockServer
TCP Server is up and running
client: k
server: You're supposed to say Knock Knock. Please try again starting with Knock Knock. Type goodbye to terminate.
client: knock, knock
server: Who's there?
```



```
Command Prompt - java TCPKnockKnockClient

knock, knock
Server Response: Who's there?

Orange
Server Response: Orange who?

Orange ya glad i didnt say banana?
Server Response: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.

Command Prompt - java TCPKnockKnockClient

Orange ya glad i didnt say banana?
Server Response: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.

k
Server Response: You're supposed to say Knock Knock. Please try again starting with Knock Knock. Type goodbye to terminate.
```

- A short description (~1 page) of the code and how to run it  
The code itself, I built the application based on the java client/server code provided in the lecture. I was able to keep track of the state of the protocol through enumerated types. I assigned each state, a number and tracked where the client was in the process that way. Every time there was an error I would throw an exception, restart the counter I to 0 (enum 0 was the knock knock step), and give the user the error message.

Open up a command prompt, cmd

Basically, the workspace is where your project is located.

```
cd $WORKSPACE
```

```
#build the class files
```

```
javac TCPKnockKnockServer.java
```

```
#run the application
```

```
java TCPKnockKnockServer
```

Open up a **second** command prompt, cmd

Basically, the workspace is where your project is located.

```
cd $WORKSPACE
```

```
#build the class files
```

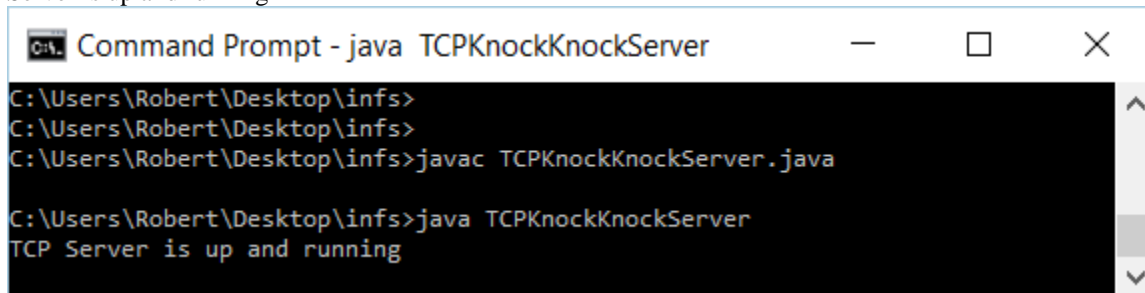
```
javac TCPKnockKnockClient.java
```

```
#run the application
```

```
java TCPKnockKnockClient
```

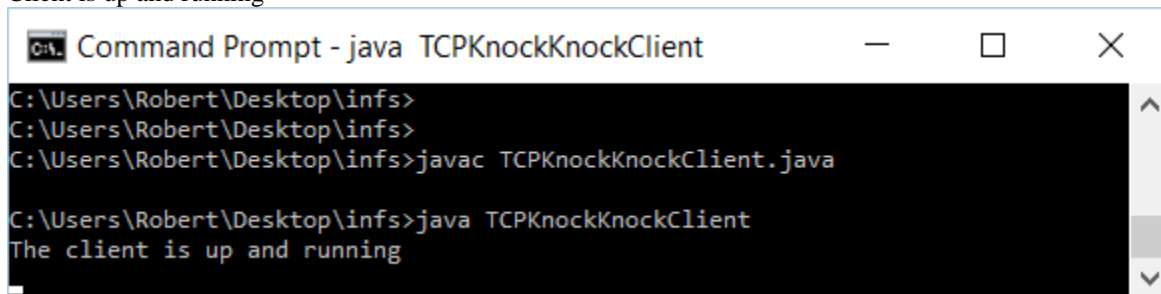
**\*\*Make sure the server is up and running before starting the client. If you follow the steps above, you'll be fine.**

Server is up and running



```
Command Prompt - java TCPKnockKnockServer
C:\Users\Robert\Desktop\info>
C:\Users\Robert\Desktop\info>
C:\Users\Robert\Desktop\info>javac TCPKnockKnockServer.java
C:\Users\Robert\Desktop\info>java TCPKnockKnockServer
TCP Server is up and running
```

Client is up and running



```
Command Prompt - java TCPKnockKnockClient
C:\Users\Robert\Desktop\info>
C:\Users\Robert\Desktop\info>
C:\Users\Robert\Desktop\info>javac TCPKnockKnockClient.java
C:\Users\Robert\Desktop\info>java TCPKnockKnockClient
The client is up and running
```

```
Command Prompt
C:\Users\Robert\Desktop\infs>
C:\Users\Robert\Desktop\infs>javac TCPKnockKnockClient.java

C:\Users\Robert\Desktop\infs>java TCPKnockKnockClient
The client is up and running
Knock Knock
Server Response: Who's there?

go
Server Response: go who?

go away
Server Response: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.

goodbye
C:\Users\Robert\Desktop\infs>
```

```
Command Prompt
C:\Users\Robert\Desktop\infs>
C:\Users\Robert\Desktop\infs>
C:\Users\Robert\Desktop\infs>javac TCPKnockKnockServer.java

C:\Users\Robert\Desktop\infs>java TCPKnockKnockServer
TCP Server is up and running

client: Knock Knock
server: Who's there?
client: go
server: go who?
client: go away
server: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.
Server shutting down
C:\Users\Robert\Desktop\infs>
```

I already described much of the code in the previous, but for the first name – last name punchline all I did was make sure the punchline started with the first name provided originally (uppercase and white space at the end ignored) and that the full punchline was greater than just the name.

For the exceptions being thrown that I mentioned earlier, I made sure that I made use of the finally block so that the error message would get read out regardless of what happens.

```
Command Prompt

C:\Users\Robert\Desktop\infs>java TCPKnockKnockClient
The client is up and running
Knock Knock
Server Response: Who's there?

joe
Server Response: joe who?

joe mama
Server Response: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.

Knock Knock
Server Response: Who's there?

Nanna
Server Response: Nanna who?

Nanna yo business
Server Response: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.

goodbye

C:\Users\Robert\Desktop\infs>_
```

```
Command Prompt

C:\Users\Robert\Desktop\infs>java TCPKnockKnockServer
TCP Server is up and running
client: Knock Knock
server: Who's there?
client: joe
server: joe who?
client: joe mama
server: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.
client: Knock Knock
server: Who's there?
client: Nanna
server: Nanna who?
client: Nanna yo business
server: HaHa. Would you like to tell another joke? If yes, please proceed starting with Knock Knock. Type goodbye to terminate.
Server shutting down

C:\Users\Robert\Desktop\infs>_
```

- A short description (~1-2 pages) of any changes you made between your protocol from Homework 1 and the protocol as implemented. You do NOT need to re-submit Homework 1 if you change your protocol, just document any changes and the reasons for them.

Quite a few things changed when I went to implement the protocol.

First thing that changed was that we weren't providing the protocol version number. This may be required for low level protocols with many versions, but here we only have 1 version. Plus the user is sending us the messages. I didn't want to require the user to send us this seemingly useless bit of information typing it in.

Another thing that changed was I allowed for the user to continue telling jokes. In the basic case, only 1 joke would be told and then the server would terminate the connection. After thinking about it I figured that the best approach would be for the client to determine when they were done telling jokes and allow for the user to terminate the connection at any time.

The last item that changed was that I decided against the use of error codes. This is a tcp connection so I figured we wouldn't have to worry about dropped packets. I would just give the error message and restart the protocol from the first step.

There were things that I would have liked to have done as well. I would have liked to create another step in the code to handle the starting the protocol over again and having the user tell us with a yes or a no if they would like to tell another joke, but it made things too complicated. It just wasn't worth it. I would just have the user start telling another joke with knock knock instead.

At first I was thinking that I would create a protocol class and have both the client and server implement that protocol class. It was just too much overhead. In the end, I just wanted the client to be as lightweight as possible and tested with the server doing all the validation.

The only other thing that I see was that I am not sending back ok messages with my who's there response. This just isn't required. All the validation is done on the server. If machines were talking to each other it would be good to have, but that's not the case here. A user is running the client instead.