

Project 4-1

Socket NetWork*

Jiashuo WANG
5100309436

April 21, 2013

Instructor: Ling Gong

I. OBJECTIVE

Write a pair of programs that will communicate over a socket, one called *AddClient* and one called *AddServer*. *AddServer* takes one command-line parameter, the port it should listen on. *AddClient* takes three command-line parameters: the address of the server, the port of the server and a string to send to the server. Each run of *AddClient* will send one string to the server.

AddServer repeatedly listens for Strings from clients. When it hears one, it attempts to convert it to an int and add it to its counter (initially 0). If the conversion fails, it just waits for the next connection. If the received String equals "exit", *AddServer* prints the value of counter to the screen and exits.

Both programs can exit on errors (cleanly, with a message), but *AddServer* must tolerate *NumberFormatException* (use try-catch block to handle it).

Format of addresses and ports:

The server will listen to a particular port on the machine where it's running. The client must connect to that port and machine combination. Thus, to test client and server, you need two command windows (if you're running on a local windows PC) or two separate terminal sessions to a remote machine (e.g. *cunix*). If you are working on a local PC, and thus both client and server will running on the same machine, you can use "*localhost*" as the address or the IP address of that machine. Choose any random port number in the range 1025-65535, for example, 2222. Then you can run the programs with `java AddServer 2222` and then `java AddClient localhost 2222`. If you are working on *cunix*, find the name of the machine where the server will run with the command `uname -n`. This may be, e.g. "walnut". So the sequence will be:

```
>uname -n
```

```
walnut
```

```
>java AddServer 2222
```

Then, in the other command window

```
>java AddClient walnut 2222
```

Or you can use *ifconfig* to find out the IP address of your machine, then replace "walnut" by the IP address.

*Designed by L^AT_EX

II. ALGORITHM

II.1 Two programs

There are two programs working together, *AddServer*, *AddClient*.

- **AddClient**

This program is used to imitate the client sending messages to the server. The main method should have three parameters, which stand for server address, server port, a string to send to the server respectively. As I work on a local PC, and thus both client and server will running on the same machine, server address will be replaced by "*localhost*". And the server port should be larger than 1024 because host number from 0 to 1024 is used for the OS. And only if the host numbers between *AddServer* and *AddClient* are the same can the communication be founded successfully. Besides, according to the rule, the sending string should be numbers, or the sending message will be ignored. That is to say, the server is like a addition machine.

The class *Socket* is important here and it offers lots of useful method to deal with the problems of socket communication.

- **AddServer**

This program is used to imitate the server receiving messages from the clients. The main method have only one parameter, server port, which must be the same with the information in client program. The class *ServerSocket* is used here to try to receive the messages.

AddServer repeatedly listens for Strings from clients. So a while-statement is needed. And in every loop a *Server* object will be created to build a connection between clients and server. As soon as the server get the message, the server program will move on and if the message is number, the counter will be increase. If the message is not number, an exception will be called and it will continue into the next loop to wait for the next message from clients.

II.2 More details

During the communication, several errors will happen. Sometimes it should cause the program to exit, sometimes, however, it should not be, especially for server. So *try – catch* statement is important to decide which error should result in the exit and which should not.

III. RESULTS AND CONCLUSIONS

III.1 Environment

- Windows 8
- NetBeans IDE 7.3

III.2 Steps

- Open the first *cmd* to start the server by using "java AddServer 2000".
- Then open the second *cmd* to start the client to send message by using like "java AddClient localhost 2000 12", and the first *cmd* will print "Roger!" and counter will increase. But if the client use "java AddClient localhost 2000 a", the first *cmd* will print "Error!" and counter will stay the same.
- After sending, the *AddClient* will exit, but the *AddServer* will keep working. Then repeat the second step to continue to send numbers.

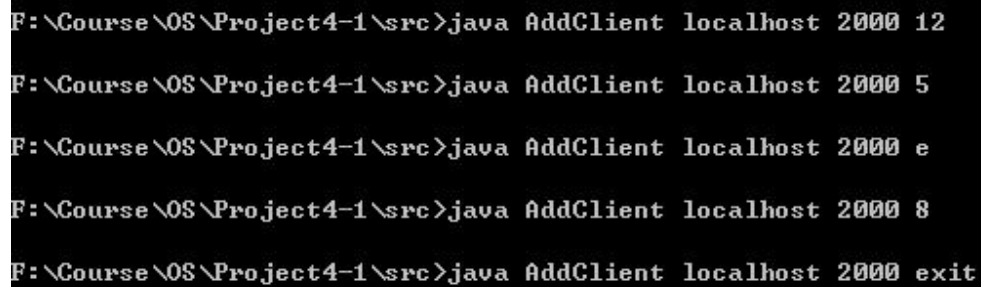
- When you want the server to exit and get the final answer, you can use "java AddClient localhost 2000 exit" in the second *cmd* and the value of counter will be printed in the first *cmd*.

III.3 Screenshots of the result

Use Command Line to compile and execute the program in Figure 1 and Figure 2.

III.4 Thoughts

As a student of EE, I think this programming skill is quite useful, and by the way, it is interesting to insult knowledge about Java in network.



```
F:\Course\OS\Project4-1\src>java AddClient localhost 2000 12
F:\Course\OS\Project4-1\src>java AddClient localhost 2000 5
F:\Course\OS\Project4-1\src>java AddClient localhost 2000 e
F:\Course\OS\Project4-1\src>java AddClient localhost 2000 8
F:\Course\OS\Project4-1\src>java AddClient localhost 2000 exit
```

Figure 1: Screenshots of AddClient



```
F:\Course\OS\Project4-1\src>java AddServer 2000
Roger!
Roger!
Error!
Roger!
counter = 25
```

Figure 2: Screenshots of AddServer