**Documentation Project 2**

**Oktaviani Szamocka s17874**

1. File

Source JDK : 1.8

This project consists of 4 java files :

1. Server

This file is for configured the length and sequences of ports for the knocking. User will fill in console. After that each port will run in thread in class ServerThread

1. ServerThread

This class is the one who open datagramsocket, listen the packet, and open the TCP connection too then listen then going to open new Thread to communicate with client in TCP connection.

1. ServerTCPThread

Sending and accepting message from client who success connect with TCP connection.

1. KnockingSequence

This class will save the right sequence that user filled into list, then when packet arrived, the client port and the port the he tried to connect will saved into map that have key the port of client and value the list of port he knocked. Later we match the ports he knocked with the right sequence.

1. Client

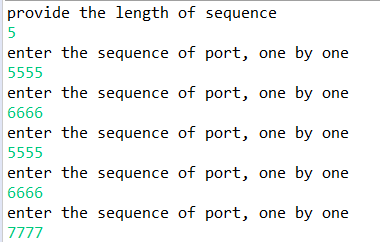
This class is the one that client going to run. There will be datagramsocket that going to send datagrampacket with destination changed according the port number filled with user in console. When knocked the right sequence, the socket will created and connected with serversocket of server. If not, in 10 second, datagramsocket will be closed.

1. Implementation

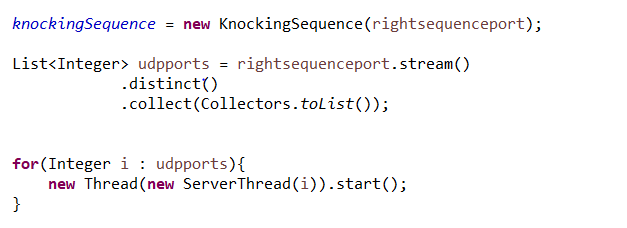
The implementation of this project assumed will happened in **localhost.**

* Implementation of right knocking

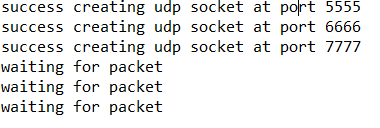
1. First of all we need to run **Server.java**
2. We r going to be asked about the length of sequence and later how the sequence will be for the right knocking. We need to fill the port one by one. See the example :



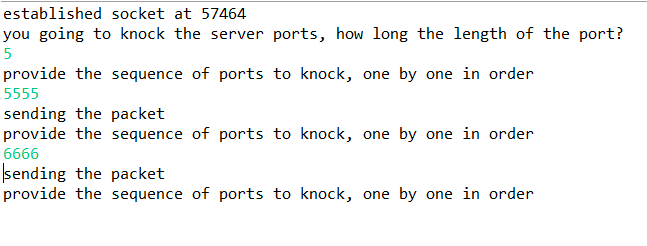
1. The sequences that we filled will be stored in *List<Integer> rightsequenceport* which will become the parameter for creating object *KnockingSequence.* And also List *rightsequenceport* will be processed by stream and distinct it (return unique value) into new List<Integer> udpports. Later each of port will be parameter of **ServerThread.java** that will be run in each thread. See the code for more understanding:



1. After that each thread of **ServerThread** class will create datagramsocket and start listening the packet that arrived.



1. Then we can run class Client.java to run the client. We will be asked also about the length and sequence of knocking. Here the example:



1. We haven’t finished the knocking, but here I want to explain the process until now. Here the client will be send the packet with destination address “localhost” and port number that user filled from console. The first packet messages is “first” . it is for indicated that it is the client first message. The rest message are “hello server”. But the last packet message is “last”.
2. From the server side, the packet that server received will be printed in console. The client socket port and the server port that he knocked will be added into **KnockingSequence** *Map<Integer, List<Integer>> sequence*. The client port as key, and server knocked will be stored in the list as the value. If it is the client first message, it will call the method remove in **KnockingSequence** to anticipate if he had ever been knocked before, so it will remove his old entry to let him try to knock again. Here the picture of server console (accepting packet from client port 57464 which knocked server with port 5555 and 6666) and code of **ServerThread.java** and **KnockingSequence.java** in this process.

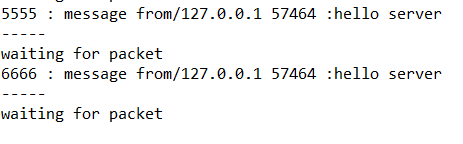


Figure 1 Server console

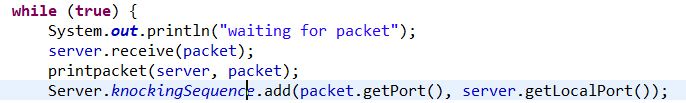


Figure 2 ServerThread code in this proces

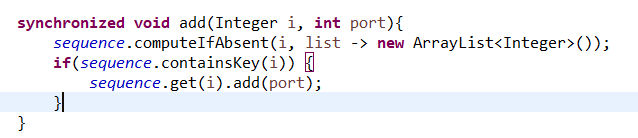


Figure 3 KnockingSequence code method add

1. Lets finish the knocking in **Client** by fill the right sequence. If client finish knocking, the last packet will have message “last” that signaled as the last packet and then server will be checked whether it s the right knocking or not. The checking process happened in **KnockingSequence** class. Remember that the *List<Integer> rightsequenceport* from **Server** class that we stored the right sequence which as parameter of creating object **KnockingSequence.** In method *equal* of **KnockingSequence** we will compared whether list of port that client knocked is equal with *List<Integer> rightsequenceport*. Here the code that illustrate this process :

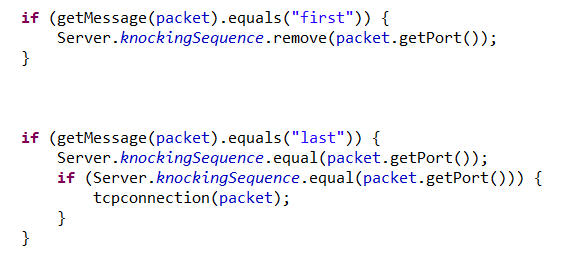


Figure 4 In ServerThread

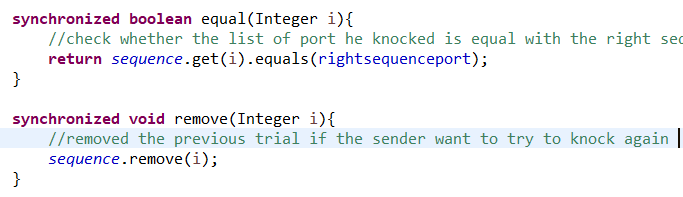


Figure 5 Method Equal in KnockingSequence

1. If it s true, the server will open TCP connection with a **ServerSocket with port 9999** and Datagramsocket will send the packet to the client with message 9999 so the client will know the port of TCP that server has opened.
2. After sending the last sequence of knocking, the client will listen the packet from server. If there is packet incoming with message contained the port number of TCP connection that server open. Client will create TCP socket with destination address “localhost” and port that mentioned before.
3. After client connected with TCP connection, client will send message “I have connected with TCP connection” through TCP socket and server will send “message accepted” after that client will close its TCP connection and its datagramsocket and being terminated. Here the picture of console if client do right knocking and got TCP connection.

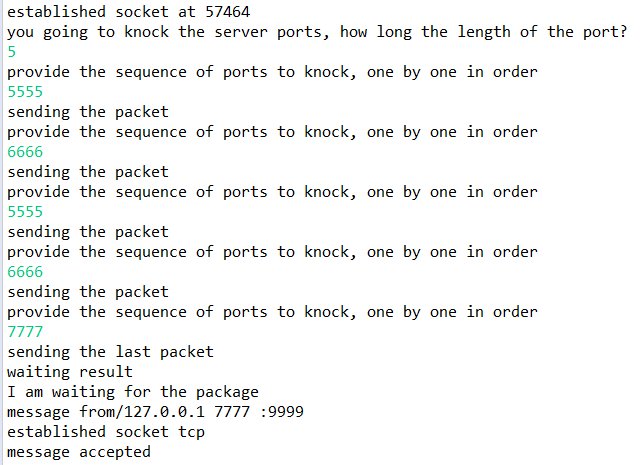


Figure 6 Client console illustration after got right sequence



Figure 7 Server console illustration after got right sequence

1. Server is keep listening the incoming client packet

* Implementation of wrong knocking

1. The process will be the same but the client has time out 10 seconds. So if there is no packet coming in 10 seconds, udp socket of client will be closed.
2. In server side if the knocking port of client is not match, server will not send anything to client, that’s why client will reach time out.

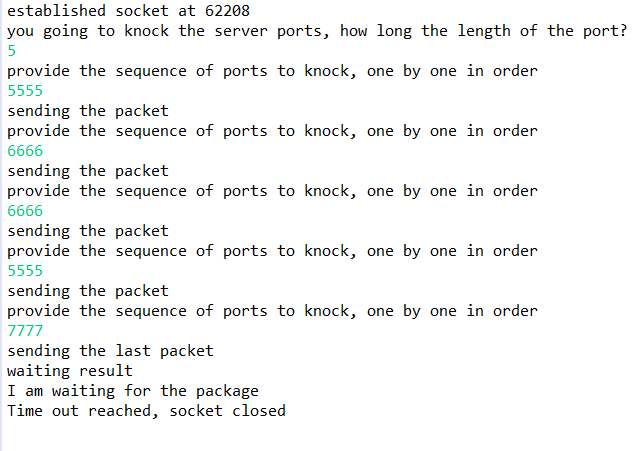


Figure 8 Client reachs time out because knocking wrong sequence

* Implementation of a few clients try to knock in the same time

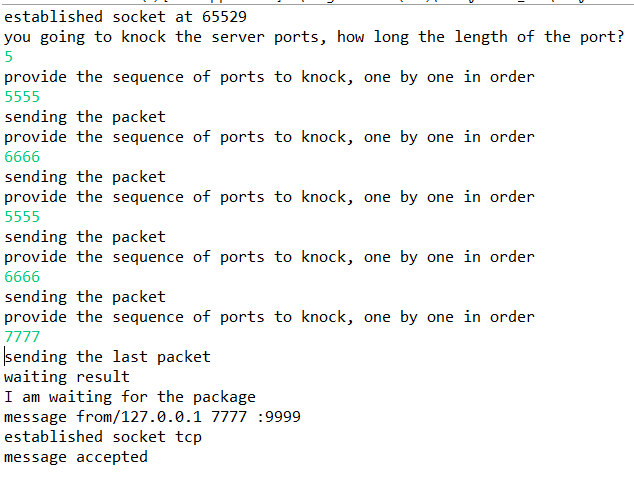
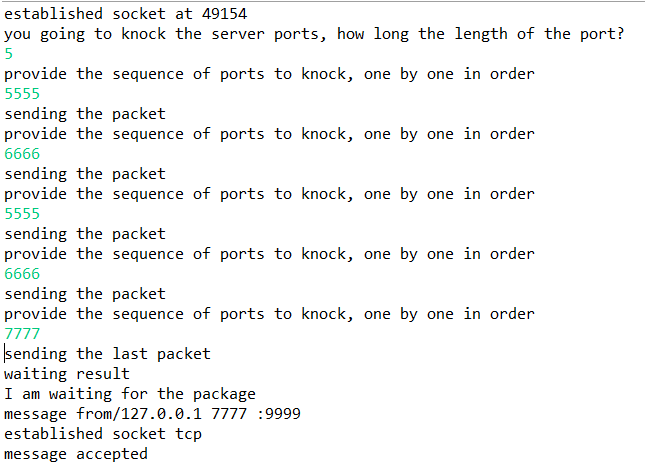
1. each datagramsocket of server run in a few threads, basically server can listen the incoming packets all the time and can process a few request of authorizing in the same time.
2. To run more than one client, you need to run client.java more than once. Here client in port 65529 and 49154 try to authorize the server in same time and both success do knocking and connected to port tcp



Figure 9 Server console illustration when 2 client try to authorize in same time