GENERAL DESCRIPTION

Project description

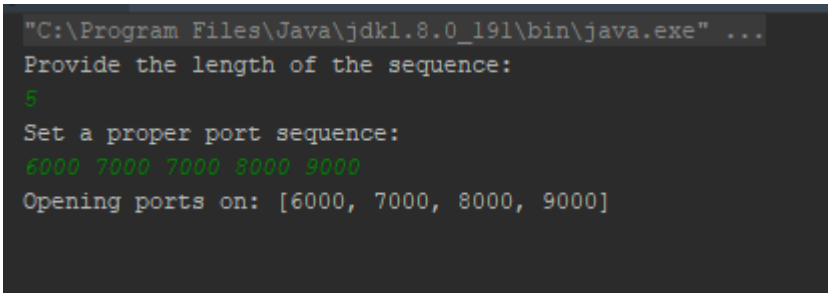
The program is written in Java SE with version JDK 1.8.

- By running the **Server** class we are asked to set the number of UDP packets to receive, and the correct order of receiving them.
- All the chosen ports are running simultaneously through objects of **MyThread** class which makes it able to listen at the same time.
- **Client** class represents the user which sets a number of ports he would like to knock and starts sending the packets to those ports.
- If packets sent by a client to the ports are correct with the sequence of ports set by a server. The TCP connection starts between them.

Example of executing the program

At the beginning we run the server and set the number of packets to be received.

After that we set the correct order of ports which should be knocked.



We got the confirmation that all the ports we want to use are opened and ready to receive packets.

Now we run the application as a client, and we have to provide the IP with which we want to connect (in our case localhost), the number packets we would like to send and the proper sequence of ports to knock.

```
"C:\Program Files\Java\jdk1.8.0_191\bin\java.exe" ...
Enter IP you want to connect to or write localhost:
localhost
Enter the length of the sequence you want to knock:
5
Provide the ports for knocking:
6000 7000 7000 8000 9000
```

The server shows the information of packets he receives, on what port and also who is the sender (we assume that packets are not lost in the network).

```
Set a proper port sequence:
6000 7000 7000 8000 9000
Opening ports on: [6000, 7000, 8000, 9000]
Port: 6000 received start message from: 127.0.0.1
Port: 7000 received middle message from: 127.0.0.1
Port: 7000 received middle message from: 127.0.0.1
Port: 8000 received middle message from: 127.0.0.1
Port: 9000 received last message from: 127.0.0.1
Correct sequence confirmed from client: 127.0.0.1!
Creating TCP connection
[TCP]: Received a request message from a client
[TCP]: Response has been sent
```

If the sequence of packets is correct then server informs that the TCP connection is being created with the given user.

```
Provide the ports for knocking:
6000 7000 7000 8000 9000
Knocking ports: 6000 7000 7000 8000 9000  on address: localhost
Accepted, starting TCP connection with the server
[TCP]: Request message has been sent to the server
[TCP]: Response message: Hello from the server
[TCP]: Connection finished
```

The client gets informed that his sequence is accepted and the TCP connection starts.

After establishing the simple request-response type communication is made, and the program terminates.

Precise description

When server starts the proper ports are stored in the LinkedList of Integers called **seqOfPorts.**

We call each distinct port inside of seqOfPorts on Threads in order to make them all listen at the same time.

We create an object of Sequence class which have 2 fields:

- LinkedList of Integers **correctSeq** which stores a proper sequence of ports set by a server
- Hashmap where the key is a object of helping class **Source** and the value LinkedList of integers **sequence**

The **Source** class represents the users. It consists of their IP and port number, the **sequence** list is the list of packets sent by this user, so we have all the information about the users trying to connect to us and also the packets sent by them.

If the server receives the packet with the message “start” it means that it’s the first packet from the user, “middle” message means that there are more packets to be received by this user, when the server finally receives the message “last” it means that all packets has been sent so we can check if the sequence from the user is equal to sequence set by a server at the beginning.

In case of meeting the requirements the server creates a TCP socket on random port and sends it to the user by UDP connection and starts to listen.

On the client side after sending all the packets, program waits for the UDP response from the server, if it arrives the TCP socket is created and connected to the server after which the simple request-response communication is made. If the client does not receive UDP response after 5 seconds the program reaches time out and terminates.

Observations

- The TCP server has to listen on the different thread which makes it possible for the port to listen for the incoming UDP packet at the same time
- The IP of the client is reset everytime when the packet with the message “start” is received in case if one client tries to connect multiple times.
- In order to make sure that the proper order of sequences is kept we use sleep() method from the class Thread after sending each packet.