Lab report Nr. 1

CHATTER BOX

	Name	Titel des Kurses	Datum
Juri Wiechmann Justin Voitel	571085 564500	Prof. Dr. Weber-Wulff Info 2 Group 2	20.10.2019

Index

Introduction

In the second Lab we started to code our first program this semester. This week we learned about the Client-Server-model and the basics about Networking in Java. We were working in groups of two. The Idea behind this was, that one is coding and meanwhile the other one is searching for ideas or helping in other ways. Every twenty minutes we switched these roles.

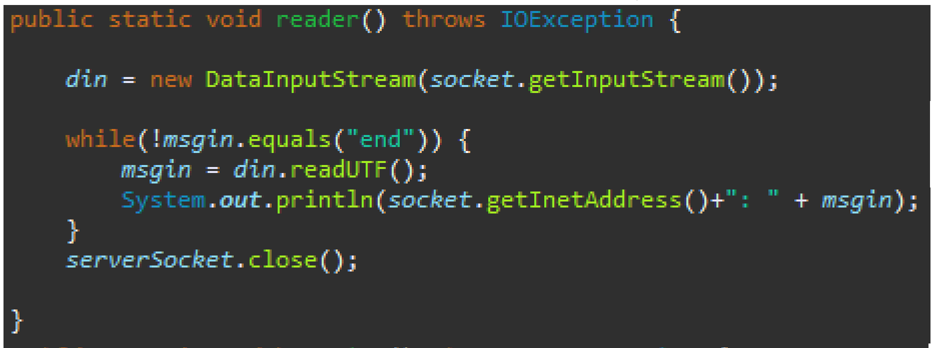
PRE-LAB

1. How do you set up an Internet connection between two computers in Java?

There are a total of 5 steps to take, when we want to set up a connection between two computers in Java

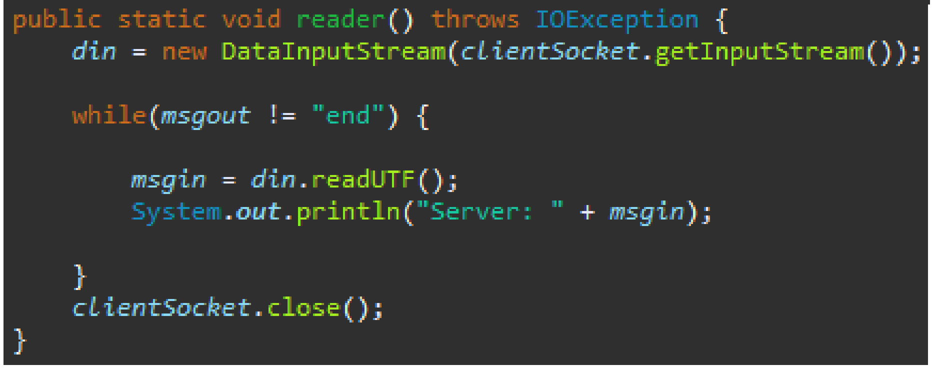
1. The first step is to open some ports which we need to set up a logical connection point.
2. The second step is to create two java-classes. One is called Client and the other one Server.
3. The third step is to create different methods. One who is reading/listening for incoming data, and the other one to write and send out data. A few more if you want but these two are necessary in each class of Client and Server.
4. Then need you can use Datainput-and Dataoutputsteams with Server sockets and Sockets that the methods do what they are supposed to do.

2. Write a method to read from a Connection in Java.



```
public static void reader() throws IOException {

    din = new DataInputStream(socket.getInputStream());

    while(!msgin.equals("end")) {
        msgin = din.readUTF();
        System.out.println(socket.getInetAddress()+": " + msgin);
    }
    serverSocket.close();

}
```

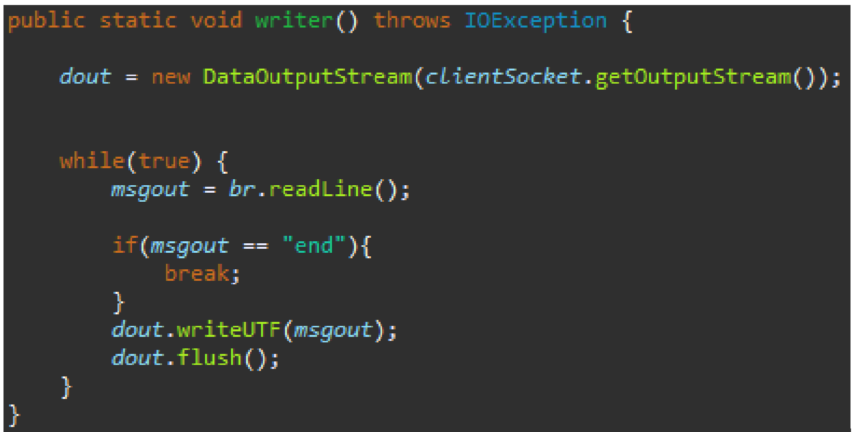
Client reader method



```
public static void reader() throws IOException {
    din = new DataInputStream(clientSocket.getInputStream());

    while(msgout != "end") {

        msgin = din.readUTF();
        System.out.println("Server: " + msgin);

    }
    clientSocket.close();
}
```

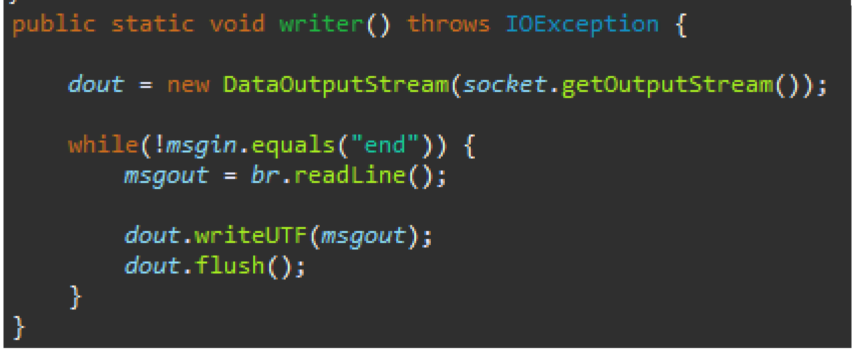
Server reader method

3. Write a method to write to a Connection in Java.



```
public static void writer() throws IOException {

    dout = new DataOutputStream(clientSocket.getOutputStream());


    while(true) {
        msgout = br.readLine();

        if(msgout == "end"){
            break;
        }
        dout.writeUTF(msgout);
        dout.flush();
    }
}
```

Client writer method



```
public static void writer() throws IOException {

    dout = new DataOutputStream(socket.getOutputStream());

    while(!msgin.equals("end")) {
        msgout = br.readLine();

        dout.writeUTF(msgout);
        dout.flush();
    }
}
```

Server writer method

Assignments

1. **Start your chatterbox by writing a method that listens on a port. The ports 8000 to 8010 are open for the lab network. This is your chatterbox server.**

The first thing we have to to is declaring our port in a static variable, to know where the clients have to set a connection. This time we will choose the port 8000.

```
static int startport = 8000;
```

Then we create a ServerSocket object with our declared port and call the accept() method to accept any connections made to this socket, which will then return a new Socket:

```
serverSocket = new ServerSocket(port);
socket = serverSocket.accept();
```

In the following we create a BufferedReader(br) that reads any input from the console from Eclipse:

```
br = new BufferedReader(new InputStreamReader(System.in));
```

Writer

So the Server is able to also write messenges to clients we first create a DataOutputStream(dout) object with the OutputStream from our accepted Sockel

```
dout = new DataOutputStream(socket.getOutputStream());
```

A while-loop will then run until the outging message reads a “end”. In that we get the message output from br using the readLine() method, which we will then write to the dout using the writeUTF() method.
The last thing this method has to do after the while-loop is clearing the dout using the flush() method:

```
while(!msgout.equals("end")) {
		msgout = br[0].readLine();

		dout[0].writeUTF(msgout);
		dout[0].flush();
}
```

Reader

In order to read messanges from the client we create a DataInputStream(din) object with the InputStream from our accepted Sockel:

```
din = new DataInputStream(socket.getInputStream());
```

A while loop that will only end if the outgoing message reads a “end” is then handling to get the message input from our din using the readUTF() method. We then want to log the input to the console extended by the clients ip address using the getInetAddress() method of our accepted Socket:

```
while(!msgout.equals("end")) {
		msgin = din.readUTF();
		System.out.println(socket.getInetAddress()+": " + msgin);
}
```

Again the last thing we want to do is closing the server socket if while loop ends:

```
serverSocket.close();
```

2. **Now write a client that writes to that port.**

We create a new Socket with the ip address of the server (e.g 172.30.80.1) and the opened port (e.g 8000) as the parameters. After that we create a BufferedReader(br) that reads any input from the console in eclipse:

Writer

For the client writer we first create a DataOutputStream(dout) object with the OutputStream from our client Sockel for writing to the server:

```
dout = new DataOutputStream(clientSocket.getOutputStream());
```

Again a while-loop will then run until the outging message reads a “end”, that will get the message output from br using the readLine() method, which we will then write to the dout using the writeUTF() method.

```
while(!msgout.equals("end")) {
        msgout = br.readLine();
        dout.writeUTF(msgout);
        dout.flush();
}
```

Reader

The clients reader will create a DataInputStream(din) object with the InputStream from our client Sockel:

```
din = new DataInputStream(clientSocket.getInputStream());
```

A while loop that will only end if the outgoing message reads a “end” is then handling to get the message input from our din using the readUTF() method. We then want to log the input to the console extended by a prefix: “Server”:

```
while(!msgout.equals("end")) {

        msgin = din.readUTF();
        System.out.println("Server: " + msgin);

}
```

Last but not least we close the client socket when the while loop ends:

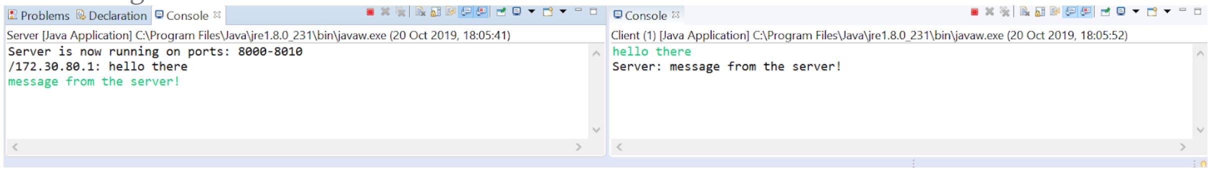
```
clientSocket.close();
```

3. **Test your methods on your own machine. For now, just echo what you have read to the console to see it working. Now publish your computer name and port number on the board in the lab.**

Our local tests were all pretty successful, we just had to open some test ports in our router configuration. We extended our Server class to store our Sockets, Streams and BufferedReader´s in an Array, so we can be able to run our Sockets on more than one Port
Our serverSockets were listening on the ports 8000-8010 and our client we run in a seperate console in eclipse that connects to the localhost ip and 8000 Port

```
clientSocket = new Socket("localhost", 8000);
```

If we now write from the client to the server or vice versa, we can see them receiving the messages



```
Server [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (20 Oct 2019, 18:05:41)
Server is now running on ports: 8000-8010
/172.30.80.1: hello there
message from the server!
```

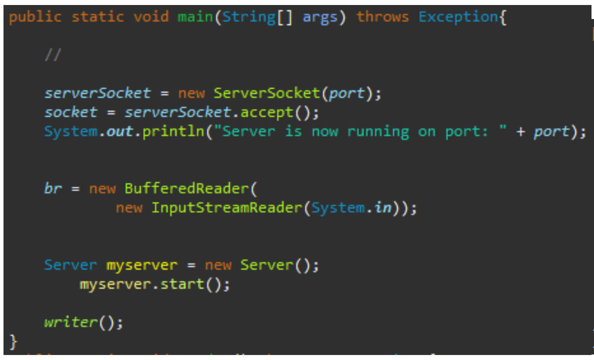
```
Client (1) [Java Application] C:\Program Files\Java\jre1.8.0_231\bin\javaw.exe (20 Oct 2019, 18:05:52)
hello there
Server: message from the server!
```

4. **Start chatting with a few of your neighbours! Describe what works and does not work.**

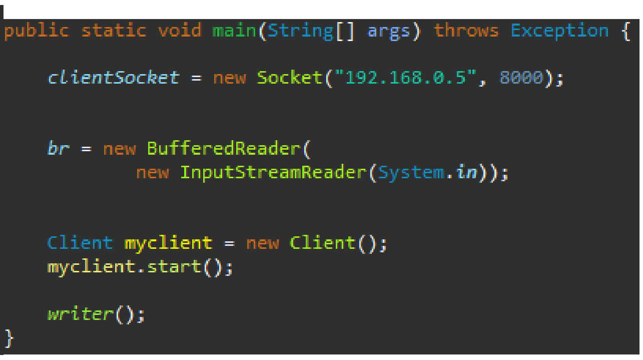
In Lab we wasn't that far to write. But we were be able to get some messages to our Server. Later we tried to improve our System that it will also write. The main problem we had was, that our write method and our read method wasn't working at the same time. We remembered about the Thread Class we was introduced in the lecture. First we had problems to think about it but after we watched a simple online video about how to implements the Thread Class and how to use the "run" method, we knew what to do.

https://www.youtube.com/watch?v=oySznjdXMEA&t

So we used the normal Thread to run the "writer" method and the extra "run" method to run the reader Method. We implements these in the Server and Client. To start these Run methods we needed to create a new Object of our Class and then to use the implemented Thread method "start()" to start the run method.

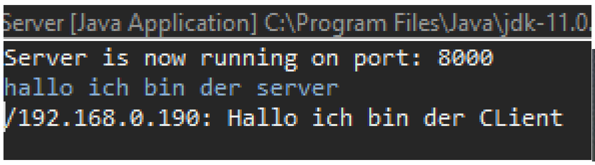


```
public static void main(String[] args) throws Exception{

    //

    serverSocket = new ServerSocket(port);
    socket = serverSocket.accept();
    System.out.println("Server is now running on port: " + port);


    br = new BufferedReader(
            new InputStreamReader(System.in));


    Server myserver = new Server();
        myserver.start();

    writer();
}
```

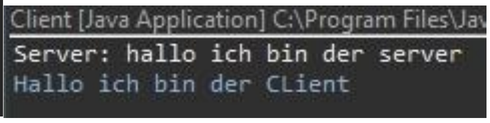


```
public static void main(String[] args) throws Exception {

    clientSocket = new Socket("192.168.0.5", 8000);


    br = new BufferedReader(
            new InputStreamReader(System.in));


    Client myclient = new Client();
    myclient.start();

    writer();
}
```

Now everything worked and one Client was be able to spam messages to the Server and the Server also to spam back without waiting for each other.



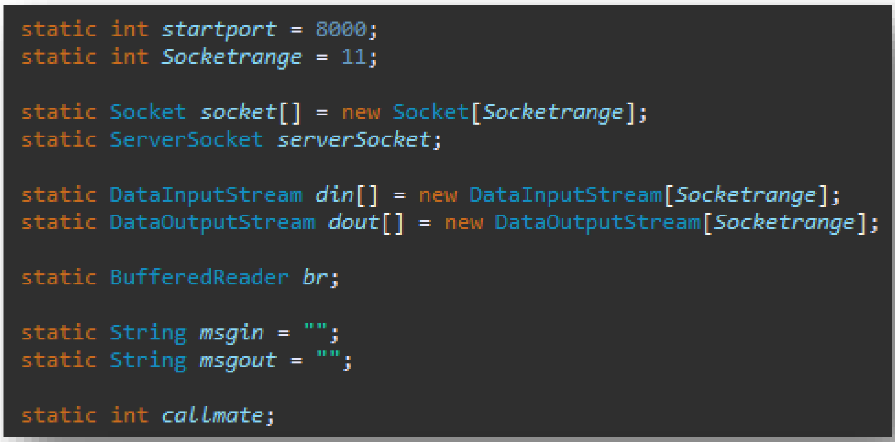
```
Server [Java Application] C:\Program Files\Java\jdk-11.0
Server is now running on port: 8000
hallo ich bin der server
/192.168.0.190: Hallo ich bin der CLient
```



```
Client [Java Application] C:\Program Files\Jav
Server: hallo ich bin der server
Hallo ich bin der CLient
```

5. **Can you get multiple clients to connect? The server will need a method broadcast that sends messages recieved to all other clients. You will need a collection for keeping track of all clients and an iterator for broadcasting.**

Our first consideration was, that we would need everything in an Array. Arrays with the length same the free socket/port-range we got. We also created an int which is called “callmate”. With this we want to dictate who we want to write if we only want to write to only one. If we want to broadcast the number will be set negative before we call the write-method. We created for every Socket we had an input- and outputstream.

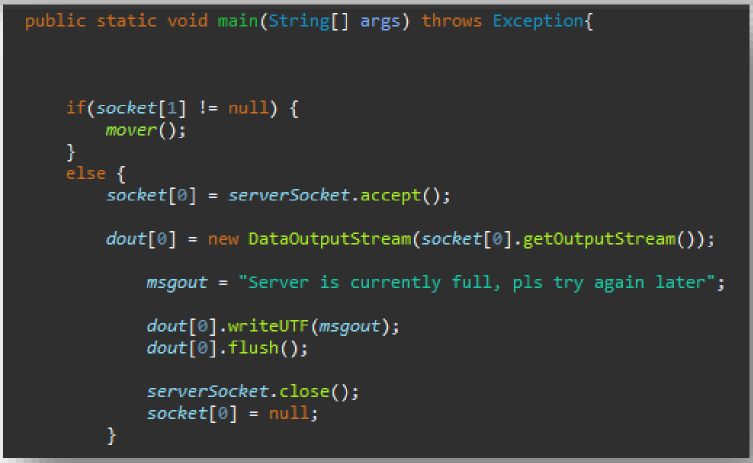


```
static int startport = 8000;
static int Socketrange = 11;

static Socket socket[] = new Socket[Socketrange];
static ServerSocket serverSocket;

static DataInputStream din[] = new DataInputStream[Socketrange];
static DataOutputStream dout[] = new DataOutputStream[Socketrange];

static BufferedReader br;

static String msgin = "";
static String msgout = "";

static int callmate;
```

Our Idea was, that our first Socket must be free all time. All Clients that want 2 connect would trigger the “move” method to find their place. We would filled up from behind and if our second socket is in use the server would be full. To make these work we also set a

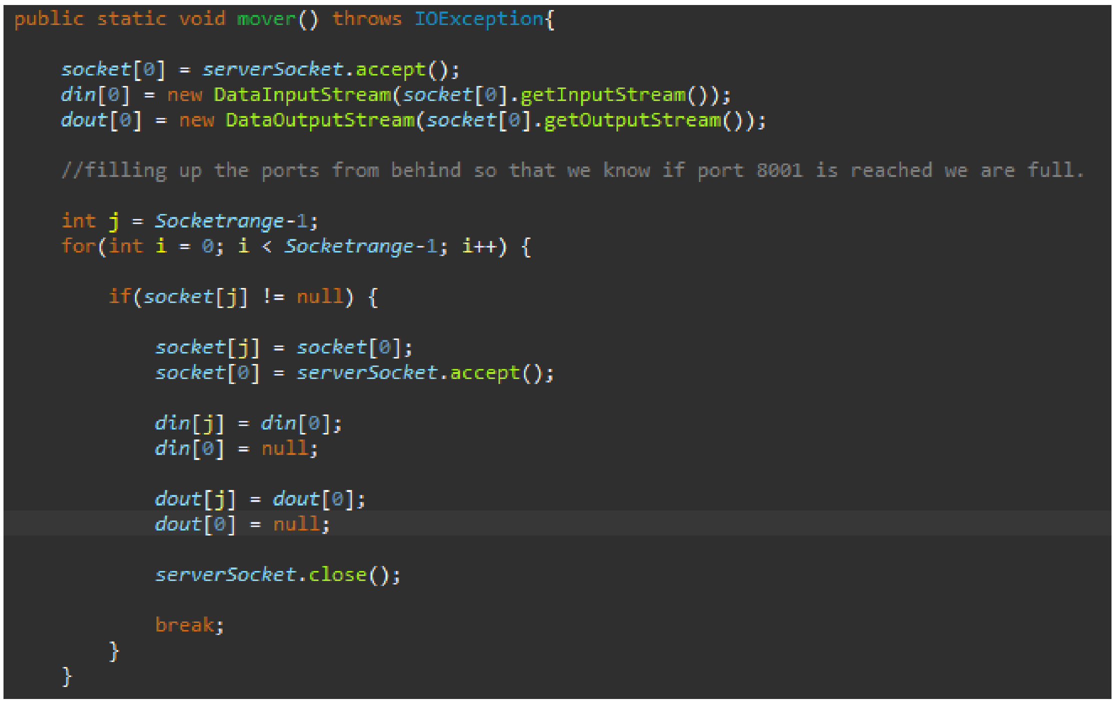
Socket[]=null;

every time a client is disconnecting



```
public static void main(String[] args) throws Exception{


    if(socket[1] != null) {
        mover();
    }
    else {
        socket[0] = serverSocket.accept();

        dout[0] = new DataOutputStream(socket[0].getOutputStream());

            msgout = "Server is currently full, pls try again later";

            dout[0].writeUTF(msgout);
            dout[0].flush();

            serverSocket.close();
            socket[0] = null;
        }
```

If our “mover”-method we set everything up that the client is connected. Now we simply searching from behind the Array a not used Socket. If we found it we switched all the streams and sockets the new Array position. After this we cleared the “entrance” streams and sockets.



```
public static void mover() throws IOException{

    socket[0] = serverSocket.accept();
    din[0] = new DataInputStream(socket[0].getInputStream());
    dout[0] = new DataOutputStream(socket[0].getOutputStream());

    //filling up the ports from behind so that we know if port 8001 is reached we are full.

    int j = Socketrange-1;
    for(int i = 0; i < Socketrange-1; i++) {

        if(socket[j] != null) {

            socket[j] = socket[0];
            socket[0] = serverSocket.accept();

            din[j] = din[0];
            din[0] = null;

            dout[j] = dout[0];
            dout[0] = null;

            serverSocket.close();

            break;
        }
    }
```

Sadly it doesn’t work. We thought about it and we came to the idea that we only call once the main method that lets the Client connect. So only the first one will come in. We came to the conclusion, that we would need a third Thread to run a Loop that constantly checks if there is a new Client asking for connection. We tried a bit but didn’t came to a solution for this. The week was over.

Reflections

JURI

Like we said at the beginning, this was my first coding experience in the lab and with the random partner and switching every 20 minutes system. Overall I think that it's a good way for people to get to know each other and learn different approaches to solving problems. On the other hand it would be nice to choose a partner with whom you got a nice working-flow. For the prelab and our first version we used a Youtubevideo. We reproduce it for have a working system. Then we tried to understand how every part of code works. We tested and changed the code. I learned, that it can be useful first to code something first, even if it is only a copy and then to understand it. While you coding I start to think about how does this peace work and you start to ask questions yourself. And if you already have these, our brain get these answer way more easily. I learned about Thread too and they are pretty useful if you need to run multiple loops or something. Sadly I couldn't figure out how to create more than 2 Threads (including the main Thread).

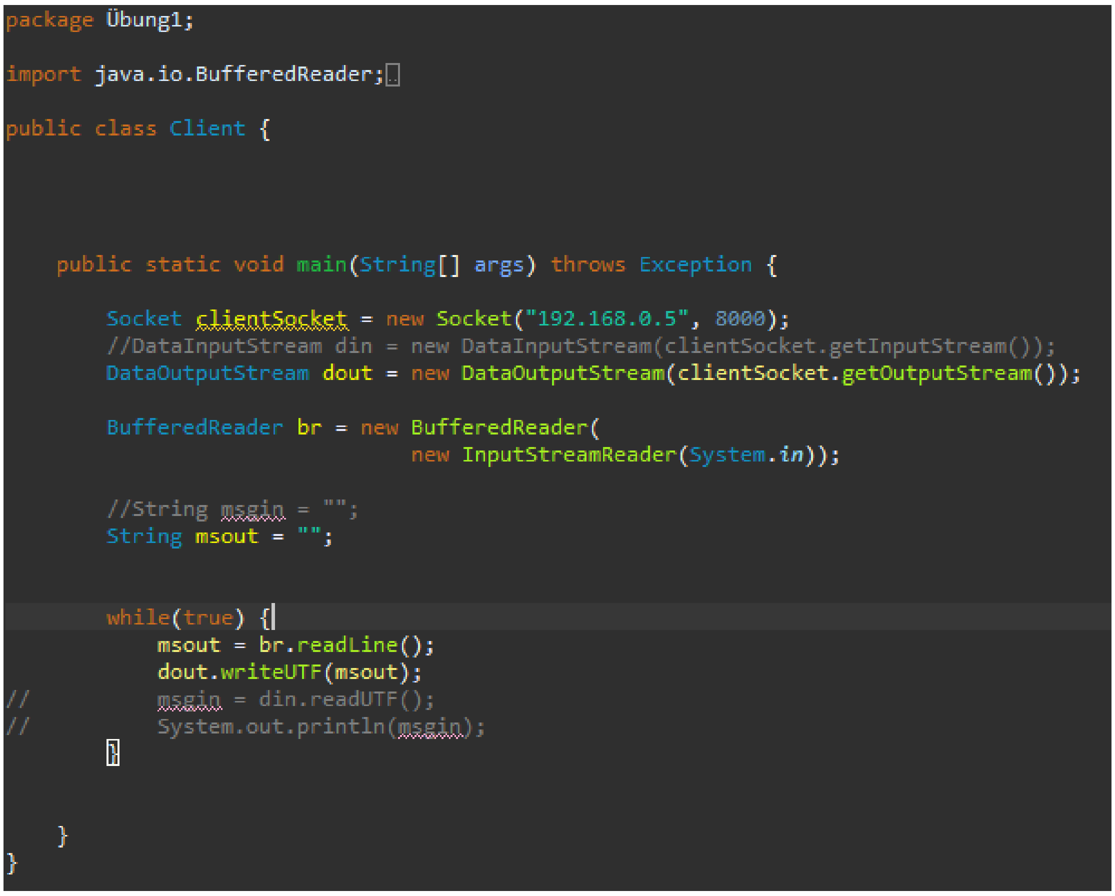
JUSTIN

This weeks lab I personally liked, because we got to know how client-Server connections are actually built. We use them every day, be it writing a message to one of our friends in facebook, the web highly depends on the client-server schema and now we know how to implement a basic socket on the client-side and a server socket on the server-side at least in Java. Iam sure this lab triggered a interest in learning more about this topic, maybe next I'll try to implement a server in JavaScript

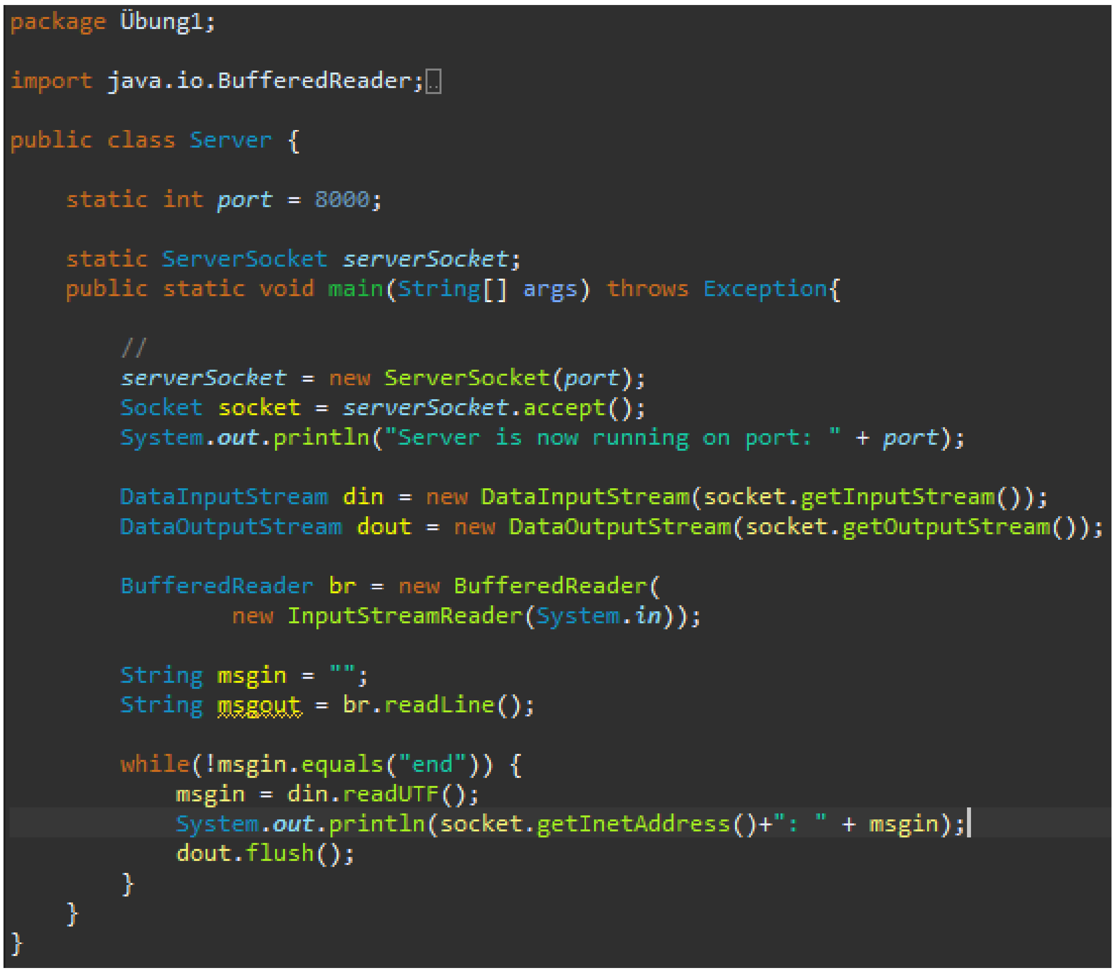
Appendix:

Java Code:

First Version(Youtube) https://www.youtube.com/watch?v=uYRTpMGdfig&t



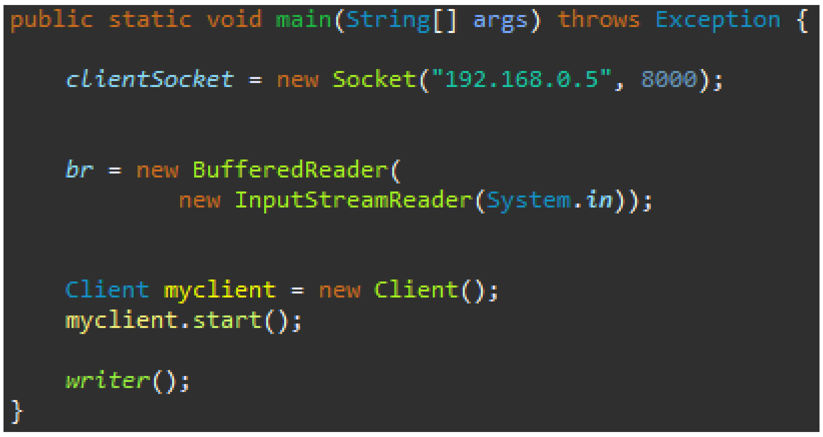
```
package Übung1;

import java.io.BufferedReader;

public class Client {



    public static void main(String[] args) throws Exception {

        Socket clientSocket = new Socket("192.168.0.5", 8000);
        //DataInputStream din = new DataInputStream(clientSocket.getInputStream());
        DataOutputStream dout = new DataOutputStream(clientSocket.getOutputStream());

        BufferedReader br = new BufferedReader(
                                new InputStreamReader(System.in));

        //String msgin = "";
        String msout = "";


        while(true) {
            msout = br.readLine();
            dout.writeUTF(msout);
//          msgin = din.readUTF();
//          System.out.println(msgin);
        }


    }
}
```

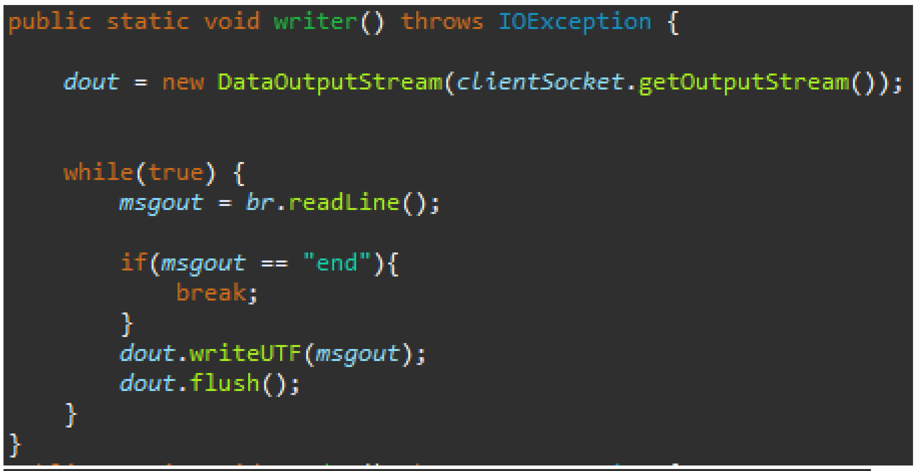
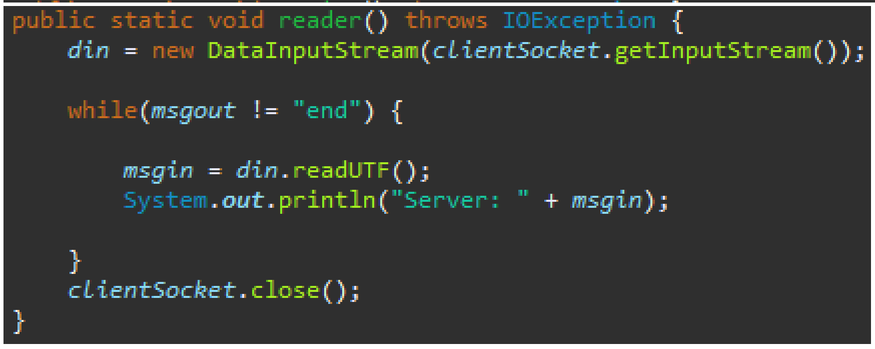

```
package Übung1;

import java.io.BufferedReader;

public class Server {

    static int port = 8000;

    static ServerSocket serverSocket;
    public static void main(String[] args) throws Exception{

        //
        serverSocket = new ServerSocket(port);
        Socket socket = serverSocket.accept();
        System.out.println("Server is now running on port: " + port);

        DataInputStream din = new DataInputStream(socket.getInputStream());
        DataOutputStream dout = new DataOutputStream(socket.getOutputStream());

        BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));

        String msgin = "";
        String msgout = br.readLine();

        while(!msgin.equals("end")) {
            msgin = din.readUTF();
            System.out.println(socket.getInetAddress()+": " + msgin);
            dout.flush();
        }
    }
}
```

Second Version:

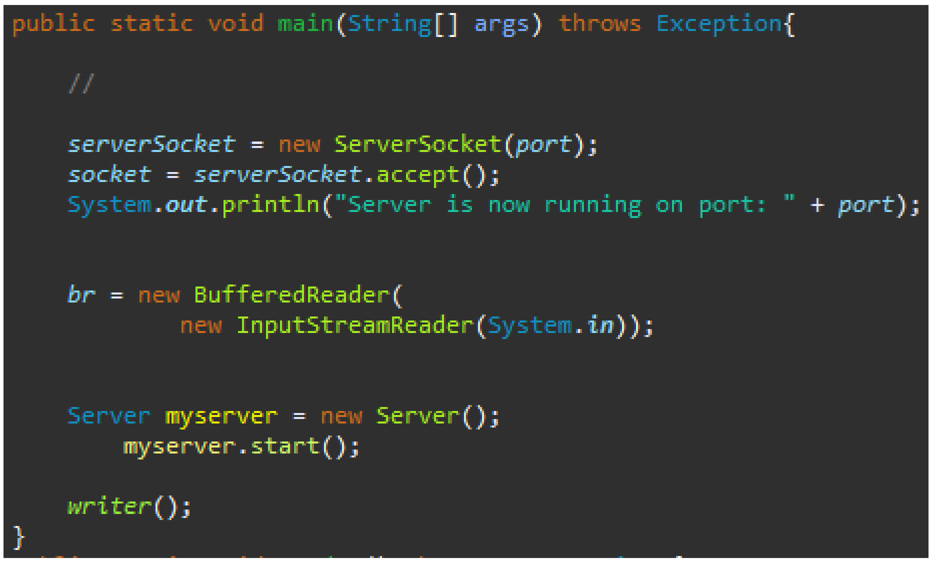
Different Methods for Write and Read and we also used Thread to run both in each Class

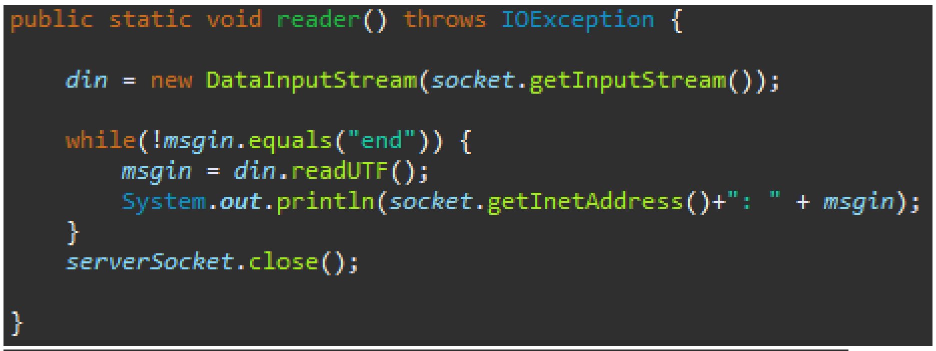
Client


```
public static void main(String[] args) throws Exception {

    clientSocket = new Socket("192.168.0.5", 8000);


    br = new BufferedReader(
            new InputStreamReader(System.in));


    Client myclient = new Client();
    myclient.start();

    writer();
}
```

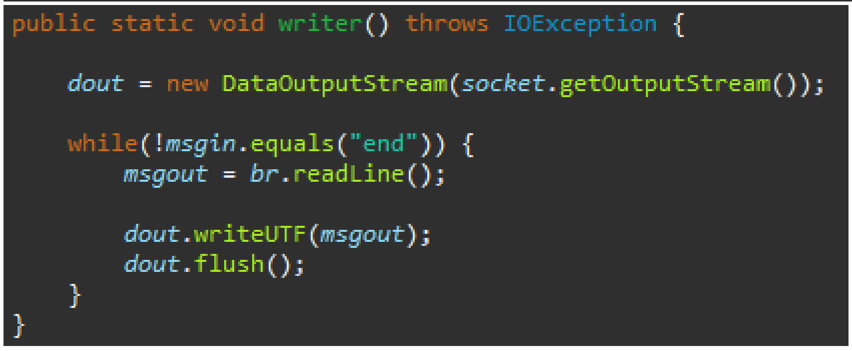

```java
public static void writer() throws IOException {

    dout = new DataOutputStream(clientSocket.getOutputStream());


    while(true) {
        msgout = br.readLine();

        if(msgout == "end"){
            break;
        }
        dout.writeUTF(msgout);
        dout.flush();
    }
}
```

```java
public static void reader() throws IOException {
    din = new DataInputStream(clientSocket.getInputStream());

    while(msgout != "end") {

        msgin = din.readUTF();
        System.out.println("Server: " + msgin);

    }
    clientSocket.close();
}
```

Server


```java
public static void main(String[] args) throws Exception{

    //

    serverSocket = new ServerSocket(port);
    socket = serverSocket.accept();
    System.out.println("Server is now running on port: " + port);


    br = new BufferedReader(
            new InputStreamReader(System.in));


    Server myserver = new Server();
        myserver.start();

    writer();
}
```

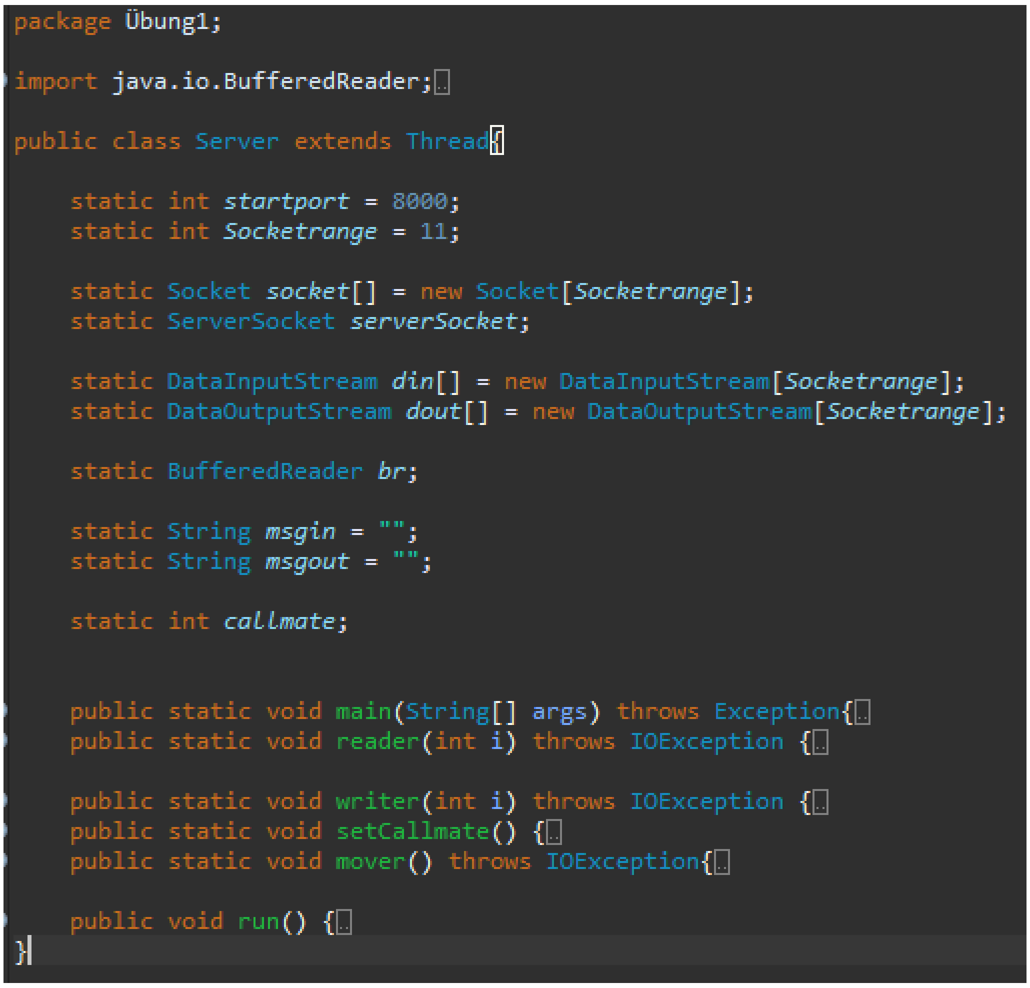


```
public static void reader() throws IOException {

    din = new DataInputStream(socket.getInputStream());

    while(!msgin.equals("end")) {
        msgin = din.readUTF();
        System.out.println(socket.getInetAddress()+": " + msgin);
    }
    serverSocket.close();

}
```

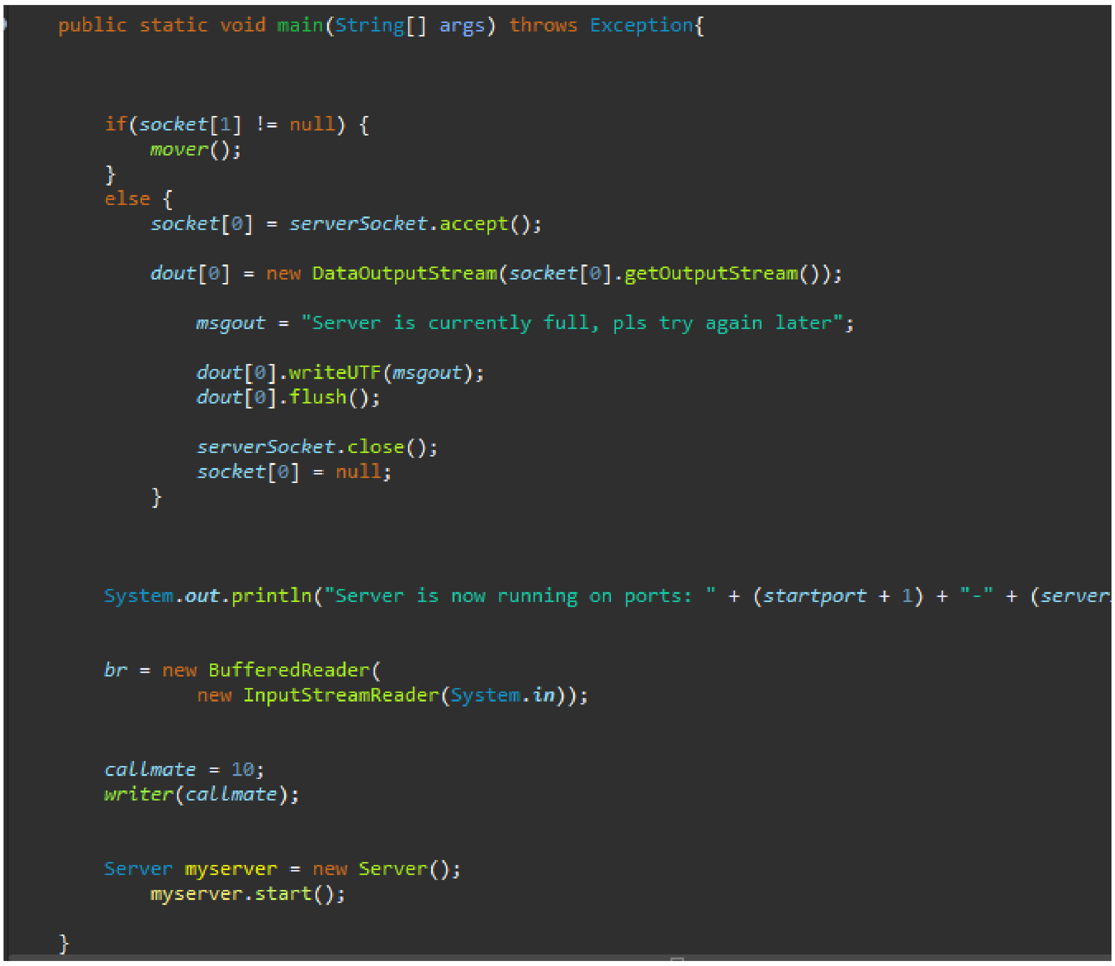


```
public static void writer() throws IOException {

    dout = new DataOutputStream(socket.getOutputStream());

    while(!msgin.equals("end")) {
        msgout = br.readLine();

        dout.writeUTF(msgout);
        dout.flush();
    }
}
```

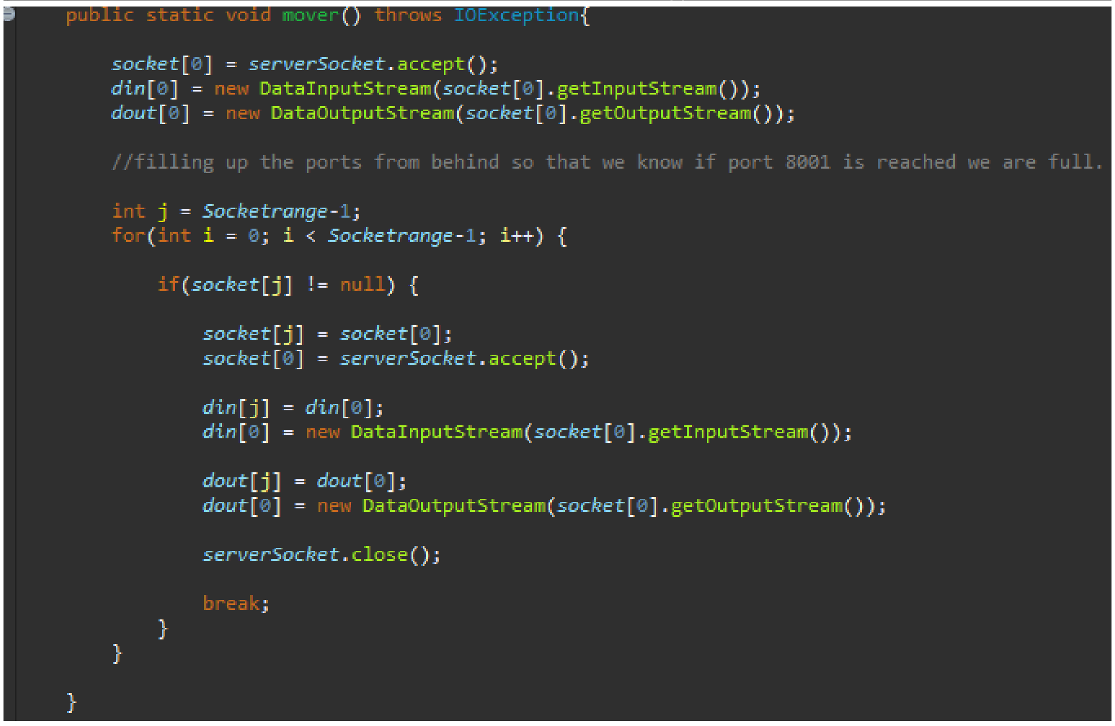
Third Version (Doesn't work) Trying to broadcast. Only Server, Client is the same like in Version two.



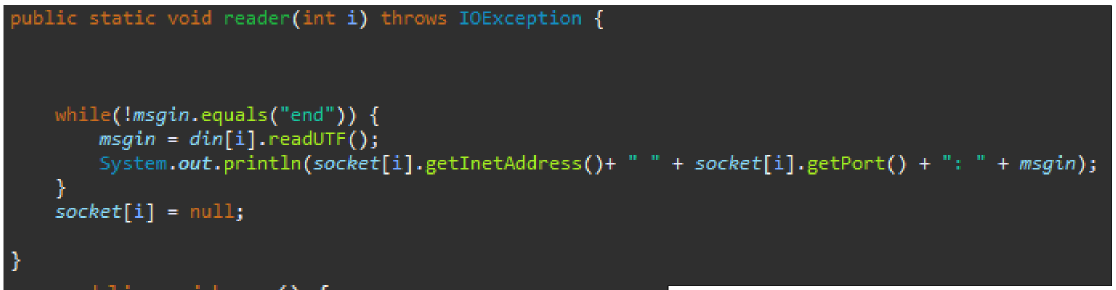
```
package Übung1;

import java.io.BufferedReader;

public class Server extends Thread{

    static int startport = 8000;
    static int Socketrange = 11;

    static Socket socket[] = new Socket[Socketrange];
    static ServerSocket serverSocket;

    static DataInputStream din[] = new DataInputStream[Socketrange];
    static DataOutputStream dout[] = new DataOutputStream[Socketrange];

    static BufferedReader br;

    static String msgin = "";
    static String msgout = "";

    static int callmate;


    public static void main(String[] args) throws Exception{
    public static void reader(int i) throws IOException {

    public static void writer(int i) throws IOException {
    public static void setCallmate() {
    public static void mover() throws IOException{

    public void run() {
}
```



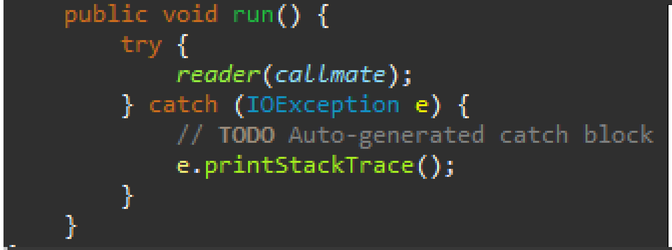
```java
public static void main(String[] args) throws Exception{


    if(socket[1] != null) {
        mover();
    }
    else {
        socket[0] = serverSocket.accept();

        dout[0] = new DataOutputStream(socket[0].getOutputStream());

            msgout = "Server is currently full, pls try again later";

            dout[0].writeUTF(msgout);
            dout[0].flush();

            serverSocket.close();
            socket[0] = null;
        }


    System.out.println("Server is now running on ports: " + (startport + 1) + "-" + (server


    br = new BufferedReader(
            new InputStreamReader(System.in));


    callmate = 10;
    writer(callmate);


    Server myserver = new Server();
        myserver.start();

}
```



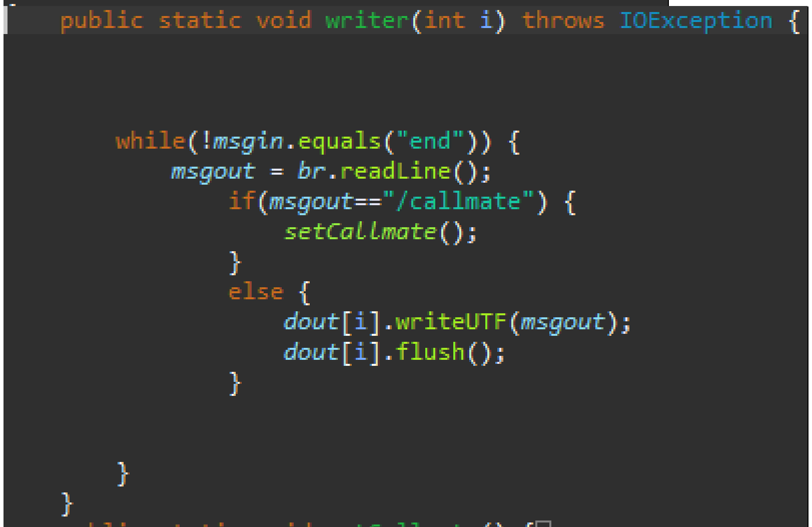
```java
public static void mover() throws IOException{

    socket[0] = serverSocket.accept();
    din[0] = new DataInputStream(socket[0].getInputStream());
    dout[0] = new DataOutputStream(socket[0].getOutputStream());

    //filling up the ports from behind so that we know if port 8001 is reached we are full.

    int j = Socketrange-1;
    for(int i = 0; i < Socketrange-1; i++) {

        if(socket[j] != null) {

            socket[j] = socket[0];
            socket[0] = serverSocket.accept();

            din[j] = din[0];
            din[0] = new DataInputStream(socket[0].getInputStream());

            dout[j] = dout[0];
            dout[0] = new DataOutputStream(socket[0].getOutputStream());

            serverSocket.close();

            break;
        }
    }

}
```



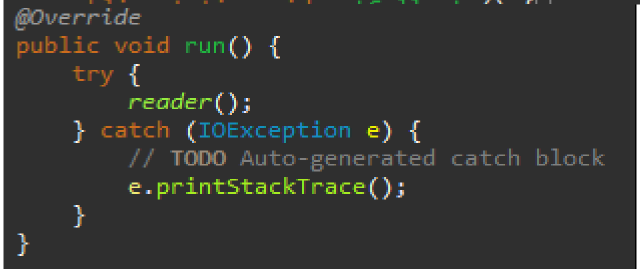
```java
public static void reader(int i) throws IOException {


    while(!msgin.equals("end")) {
        msgin = din[i].readUTF();
        System.out.println(socket[i].getInetAddress()+ " " + socket[i].getPort() + ": " + msgin);
    }
    socket[i] = null;

}
```



```java
    public void run() {
        try {
            reader(callmate);
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```



```java
    public static void writer(int i) throws IOException {



        while(!msgin.equals("end")) {
            msgout = br.readLine();
                if(msgout=="/callmate") {
                    setCallmate();
                }
                else {
                    dout[i].writeUTF(msgout);
                    dout[i].flush();
                }


        }
    }
```



```java
@Override
public void run() {
    try {
        reader();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```