

Students : Lena Zellin
Pavel Tsvyatkov

Email : s0571015@htw-berlin.de
s0559632@htw-berlin.de

Instructor: Prof. Dr. Debora Weber Wulff

Laboratory Report

Exercise 1: Chatterbox

Lab-plan

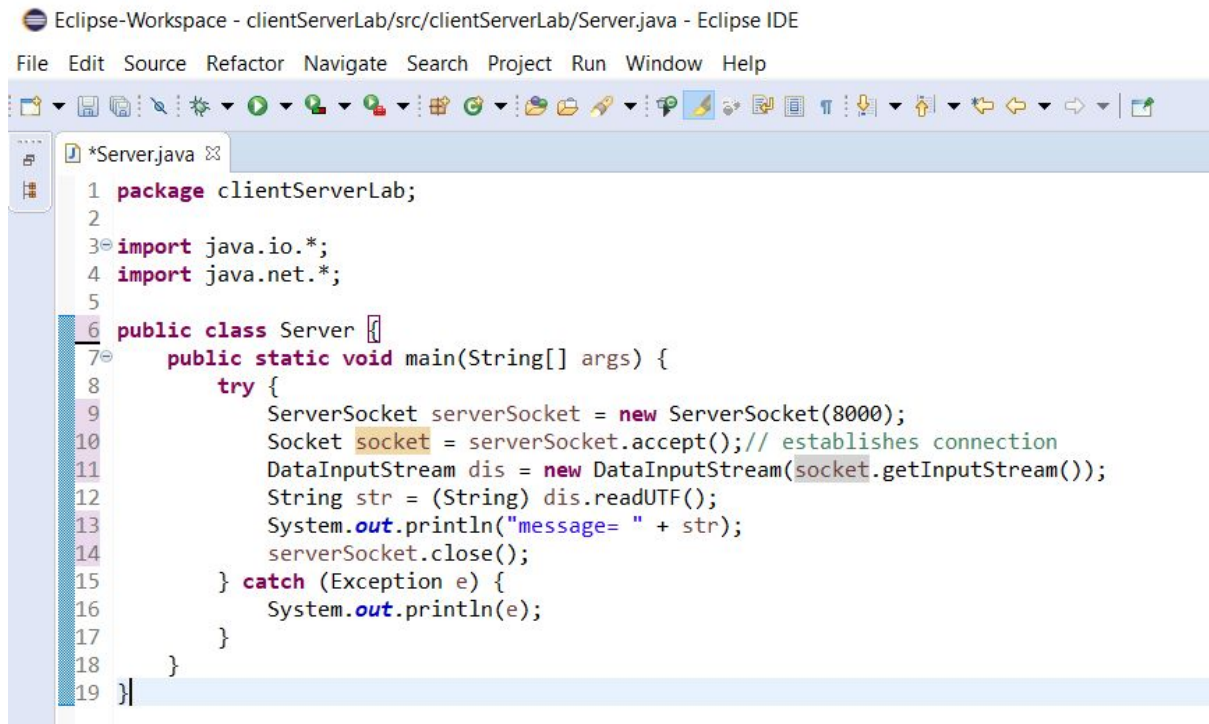
This lab was about creating a Server and Client to chat and exchange simple messages with other lab members.

Assignment

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.

Before we began with the exercise, we discussed what we have found so far about socket programming while doing the finger exercises. We were both unsure if we had found the correct methods while researching, but we just decided to try and see what works and what doesn't. We decided to work in Eclipse, so we created a new Java project and created a new class called Server. We found some information on how socket programming works on the following site: <https://www.javatpoint.com/socket-programming>. There was a very well made and explained diagram about how the Client and the Server communicate and there were methods with examples and explanations, which we found very useful. Before moving on to try it in Eclipse, we read through the lines of code step by step and made sure we understand it. We had to change the

ServerSocket and make it listen to one of the opened ports in the Lab, so in the Server class we changed it to listen to port 8000.



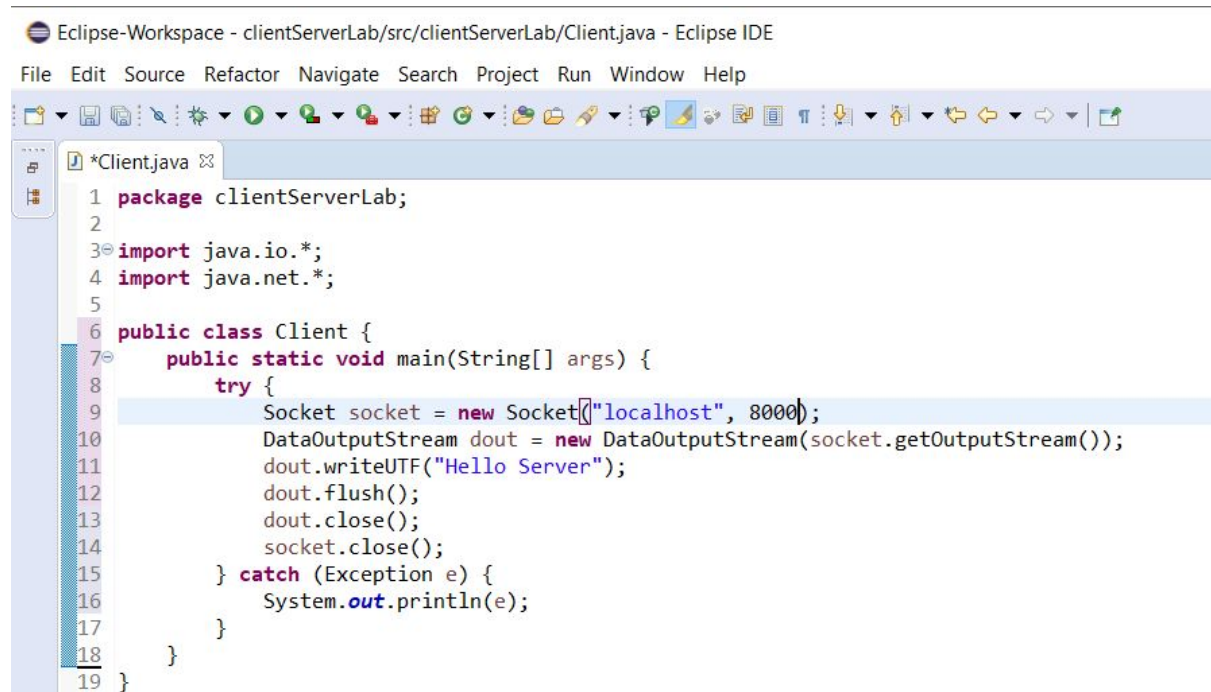
```
Eclipse-Workspace - clientServerLab/src/clientServerLab/Server.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

1 package clientServerLab;
2
3 import java.io.*;
4 import java.net.*;
5
6 public class Server {
7     public static void main(String[] args) {
8         try {
9             ServerSocket serverSocket = new ServerSocket(8000);
10            Socket socket = serverSocket.accept();// establishes connection
11            DataInputStream dis = new DataInputStream(socket.getInputStream());
12            String str = (String) dis.readUTF();
13            System.out.println("message= " + str);
14            serverSocket.close();
15        } catch (Exception e) {
16            System.out.println(e);
17        }
18    }
19 }
```

At this point we weren't sure if that was the correct way to do it, but we didn't have any other idea at the moment so we continued with writing the Client class.

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

We created a new Client class that is supposed to write to our Server. We weren't sure if it's possible to make it work, because we were trying to make the machine be Client and a Server at the same time. We decided to try the code from the site with explanation we found and put it in the Client class. We changed the port to 8000 and left localhost as it was.

A screenshot of the Eclipse IDE interface. The title bar reads 'Eclipse-Workspace - clientServerLab/src/clientServerLab/Client.java - Eclipse IDE'. The menu bar includes 'File', 'Edit', 'Source', 'Refactor', 'Navigate', 'Search', 'Project', 'Run', 'Window', and 'Help'. The toolbar contains various icons for file operations, editing, and running. The editor window shows the code for 'Client.java'. The code is as follows:

```
1 package clientServerLab;
2
3 import java.io.*;
4 import java.net.*;
5
6 public class Client {
7     public static void main(String[] args) {
8         try {
9             Socket socket = new Socket("localhost", 8000);
10            DataOutputStream dout = new DataOutputStream(socket.getOutputStream());
11            dout.writeUTF("Hello Server");
12            dout.flush();
13            dout.close();
14            socket.close();
15        } catch (Exception e) {
16            System.out.println(e);
17        }
18    }
19 }
```

We had no idea if that was the correct way to write the two classes so we asked Professor Weber-Wulff about it. She told us that we should be implementing the Runnable class and have a method run() that takes care of the process to be constantly running and that we should be reading input from the keyboard(System.in). She also suggested that we should probably have a few System.outs that say when the Server is started and when a message is sent from a Class. We also knew that we should have a while loop that loops until we say a certain thing, but we didn't have an idea how to structure our code at this moment. We were trying different things, but it was either not working or there was no change.

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.

At this point we knew the code worked and how we could improve it, according to our Professor. We started with implementing a way to read from the keyboard, because as off right now, we only input a fixed statement "Hello Server" that gets send and shown in the console. This didn't really satisfy us, so we both talked about it and decided that we would try and redo the code/come up with our own as good as we could, so we can input whatever and have that

send to the Server. We then started revising our code at home and came up with this for the Client:

```
1
2 import java.io.BufferedReader;
3
4
5
6
7
8
9 class Client implements Runnable {
10     Socket socket;
11
12     public static void main(String[] args) {
13         new Thread(new Client()).start();
14     }
15
16     @Override
17     public void run() {
18         try {
19             socket = new Socket("localhost", 51201);
20             BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
21             DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream());
22
23             System.out.println("Write a message...");
24
25             while(true) {
26                 String inputString = reader.readLine();
27                 outputStream.writeUTF(inputString);
28                 outputStream.flush();
29
30                 if (inputString.equals("bye")) {
31                     break;
32                 }
33             }
34
35             socket.close();
36             reader.close();
37             outputStream.close();
38         }
39         catch (Exception e) {
40             System.out.println(e);
41         }
42     }
43 }
44 }
```

Our Client has one field that is of type Socket, which we just named socket.

We asked Professor Weber-Wulff about this beforehand and she told us our classes would both need to implement Runnable as stated above. This means that both classes need to have a run method of the Thread class that exists in Java.

The run method in Client creates a new Socket that gets the servers address as a String, we put "localhost" there to connect to our own Server class later and then an int port to which to connect to. A lot of ports over 49000 are pretty much free to use so we put 51201 randomly there.

Next we create a BufferedReader, that implements a new InputStreamReader, which reads from System.in, the users keyboard. Then we needed a DataOutputStream, which gets the OutputStream of our Socket.

At first we didn't use `DataOutputStream` for reasons we can't remember, but then Lena asked Marie and Katja after class what they had used and they said `DataOutputStream`, which made a lot of sense, so we used that as well.

We added a `System.out.println("Write a message...");` to tell the user to start typing once Server and Client are connected.

We then start a while loop. The while loops condition is `true`, because we want the Server and Client connection to keep running and running and do the following things over and over:

We create a `String` that consists of whatever we typed into `System.in` that gets then input by our `InputStreamReader` and then read and processed by our `BufferedReader`. The `BufferedReader` reads the line of whatever we input, basically.

Our `DataOutputStream` then writes this `String` using the method `writeUTF`, which simply writes a `String` to an `OutputStream`. After that we flush our `DataOutputStream`, so the data gets send out immediately.

After that comes an important part: we used an if-statement, which checks whether or not the `String` we typed in was `"bye"`. If it was we break out of our while loop and close our `Socket`, our `BufferedReader` and our `DataOutputStream`. If we didn't type `bye` the loop just goes on.

All of this is surrounded by a try and catch block, since `BufferedReader`, for example, requires a handling for `IOException`, so we decided to put all of our run method code into the try segment. This doesn't run on it's own however.

```

7  class Server implements Runnable {
8      ServerSocket serverSocket;
9
10 public static void main(String[] args) {
11     new Thread(new Server()).start();
12 }
13
14 @Override
15 public void run() {
16
17     try {
18         serverSocket = new ServerSocket(51200);
19         Socket readerSocket = serverSocket.accept();
20         DataInputStream inputReader = new DataInputStream(new BufferedInputStream(readerSocket.getInputStream()));
21
22         while (true) {
23             String messageString = inputReader.readUTF();
24
25             if (messageString.equals("bye")) {
26                 System.out.println("Bye!");
27                 break;
28             } else {
29                 System.out.println("Client said: " + messageString);
30             }
31         }
32
33         readerSocket.close();
34         serverSocket.close();
35         inputReader.close();
36
37     } catch (IOException e) {
38         System.out.println(e);
39     }
40 }
41 }
42 }
43 }

```

Our Sever class has a field of type ServerSocket, which is a Socket that waits for a connection. This class also implements Runnable like our Client class.

The run method has a try and catch block as well. In the try segment we create a new ServerSocket, which listens on port 51201 and then we create a Socket that then waits at the ports where the ServerSocket is and waits for the ServerSocket to connect to something using the accept method.

We then need a DataInputStream, which creates a BufferedInputStream that gets the input Stream of our Socket.

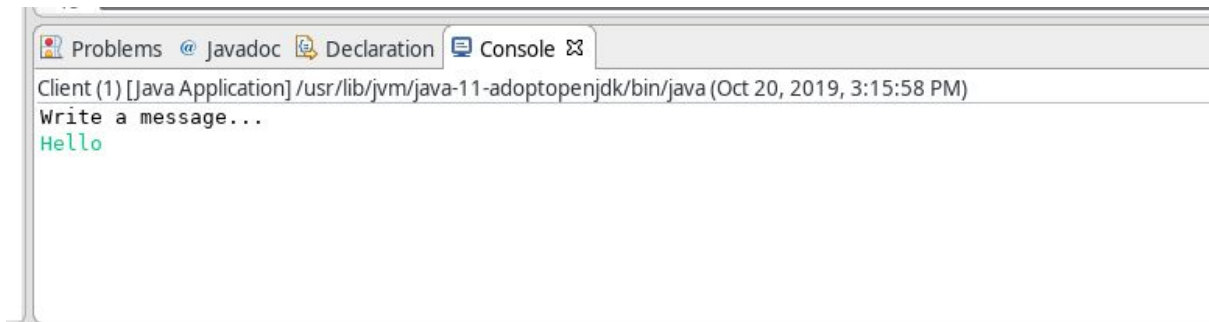
Then we have a while loop again that's always true and we create a second String that we get from reading the input data with the readUTF() method.

We then check if the message contains "bye" and if it does the Server prints "Bye!" to the console, showing that the conversation is over. Then we break out of the loop and close our ServerSocket, our Socket and the DataInputStream.

If it doesn't contain "bye" we print out "Client said : " plus the messageString so we know what and that the Client wrote something.

Both of our main methods create an object Thread in which we create either a Client or Server object, according to the class and then calling the start method. The start method is a method that belongs to the Thread class, and what it does is basically starting the run method that we implemented and keeping it running.

We then ran our Sever and our Client, and they both connected!



```
Problems @ Javadoc Declaration Console
Client (1) [Java Application] /usr/lib/jvm/java-11-adopopenjdk/bin/java (Oct 20, 2019, 3:15:58 PM)
Write a message...
Hello
```

Client writing and sending a message



```
Problems @ Javadoc Declaration Console
Server (1) [Java Application] /usr/lib/jvm/java-11-adopopenjdk/bin/java (Oct 20, 2019, 3:15:56 PM)
Client said: Hello
```

Server receiving the message



```
Problems @ Javadoc Declaration Console
<terminated> Client (1) [Java Application] /usr/lib/jvm/java-11-adopopenjdk/bin/java (Oct 20, 2019, 3:15:58 PM)
Write a message...
Hello
bye
```

Client writing and sending bye

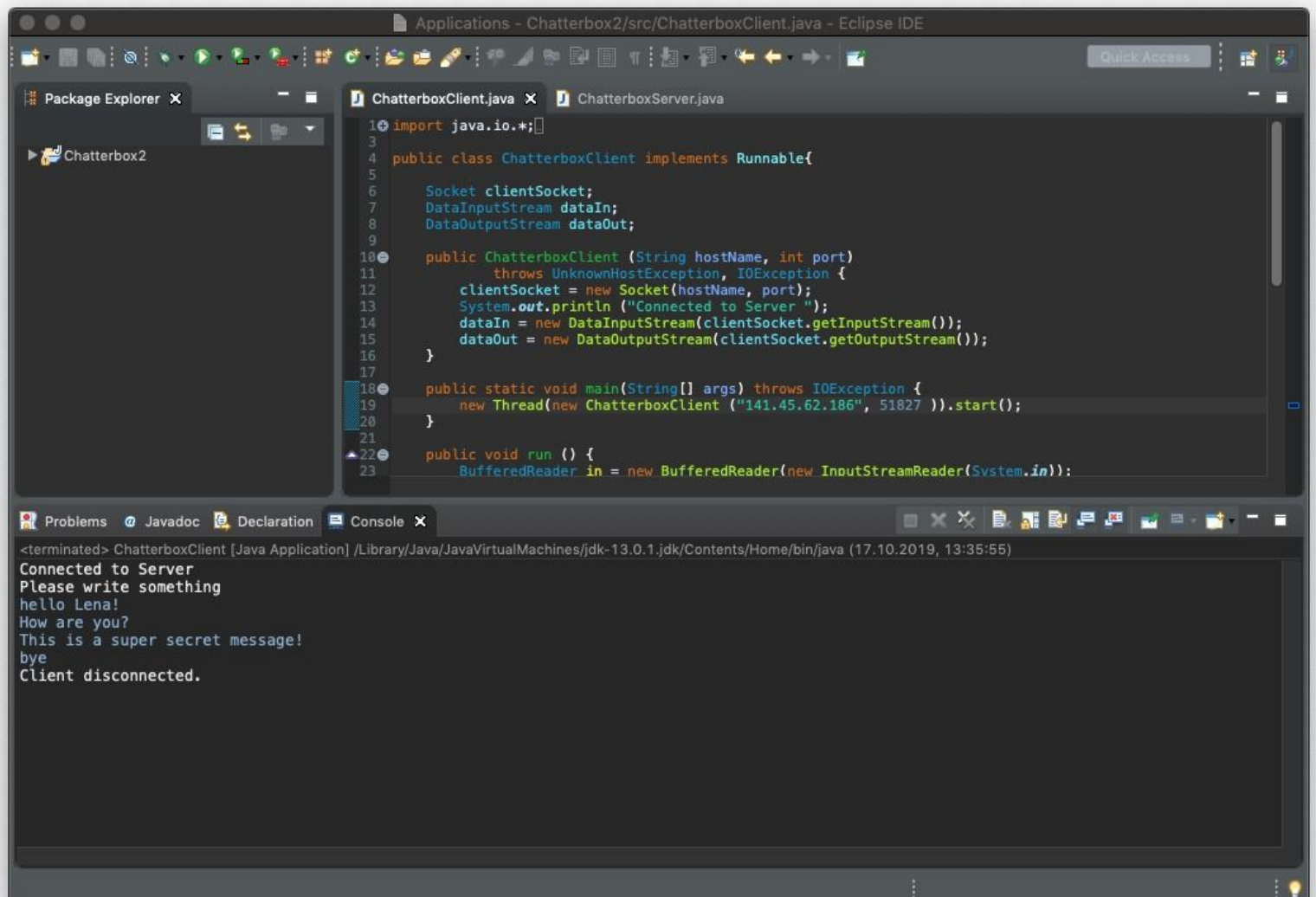


Server ending the connection

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

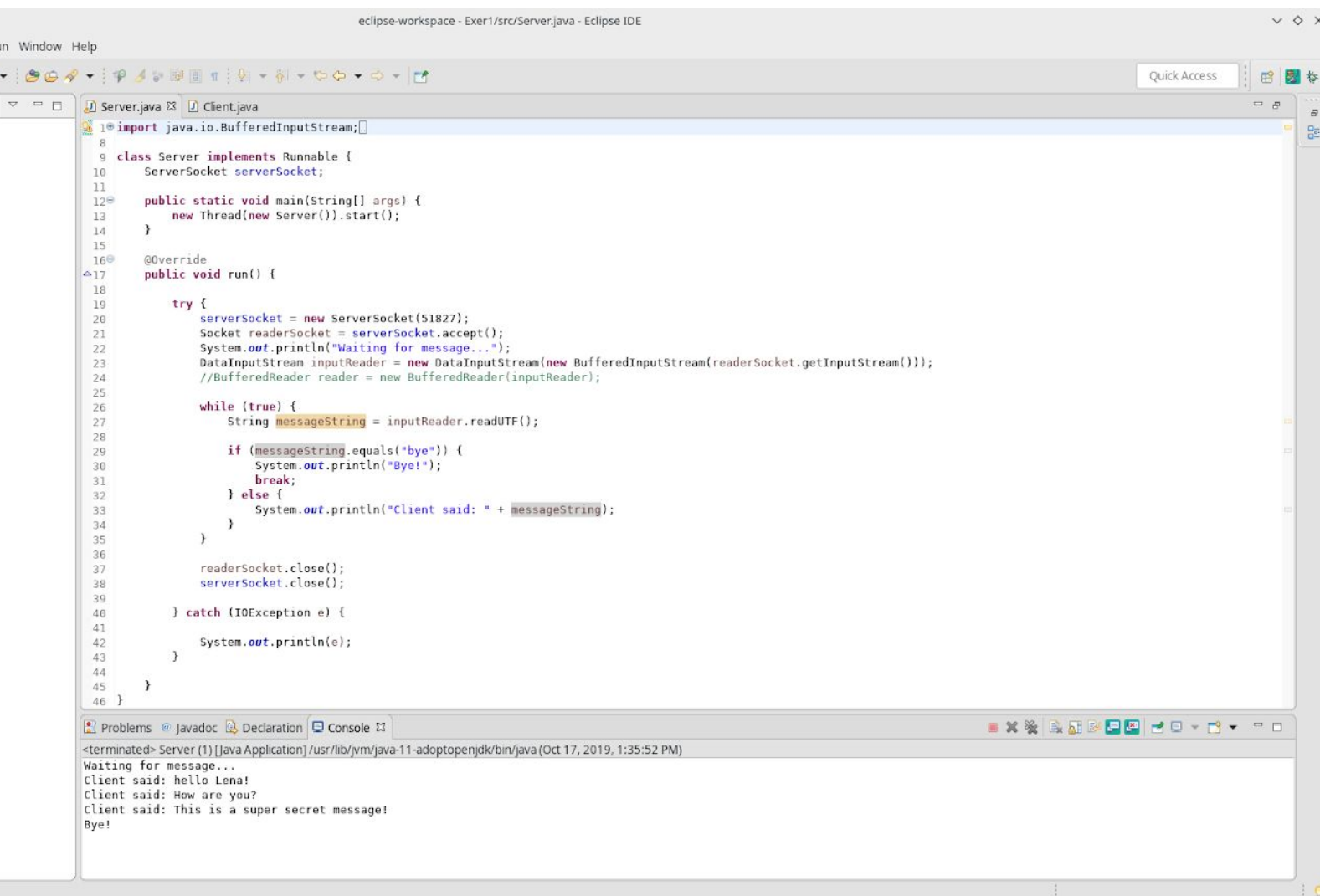
After we saw that this worked we decided to go and connect to someone else. As we already collaborated with them before, we decided to ask Katja and Marie again, if they wanted to test their Client and Server classes with us.

First we were the Server and the other group was the Client. We gave them our port and IP address. They connected and they could send us messages :



Maries and Katjas Client class writing

We then tried it the other way around, which also worked and had the exact same outcome.



The screenshot shows the Eclipse IDE with the `Server.java` file open. The code implements a `Runnable` class that listens for client connections on port 51827. It uses `ServerSocket`, `Socket`, `InputStreamReader`, and `BufferedReader` to handle the communication. The `run()` method enters a loop where it reads messages from the client and prints them to the console. It also handles a "bye" message by breaking the loop and closing the sockets.

```
1 import java.io.*;
2
3 class Server implements Runnable {
4     ServerSocket serverSocket;
5
6     public static void main(String[] args) {
7         new Thread(new Server()).start();
8     }
9
10    @Override
11    public void run() {
12        try {
13            serverSocket = new ServerSocket(51827);
14            Socket readerSocket = serverSocket.accept();
15            System.out.println("Waiting for message...");
16            InputStreamReader inputReader = new InputStreamReader(new BufferedInputStream(readerSocket.getInputStream()));
17            //BufferedReader reader = new BufferedReader(inputReader);
18
19            while (true) {
20                String messageString = inputReader.readUTF();
21
22                if (messageString.equals("bye")) {
23                    System.out.println("Bye!");
24                    break;
25                } else {
26                    System.out.println("Client said: " + messageString);
27                }
28            }
29
30            readerSocket.close();
31            serverSocket.close();
32        } catch (IOException e) {
33            System.out.println(e);
34        }
35    }
36 }
```

The console output shows the server's execution:

```
<terminated> Server (1) [Java Application] /usr/lib/jvm/java-11-adopptopenjdk/bin/java (Oct 17, 2019, 1:35:52 PM)
Waiting for message...
Client said: hello Lena!
Client said: How are you?
Client said: This is a super secret message!
Bye!
```

Our Server receiving the other groups messages

The only thing that doesn't work with our implementation is answering as a Server. We can't write back, which may not qualify as "chatting" if we mean a two way conversation.

For some reason, the idea to connect our Server with their Client and our Client with their Server at the same time didn't cross our mind, which would have

qualified as chatting. Seeing as both connections independently worked, we think that connecting them up to really chat would have worked though.

Also, we never normalize the String which we input as the Client and because of that, "bye" is the only String that ends the connection to the Server, not "BYE" or any other spelling.

Personal Reflection

Lena: I personally had a hard time with this lab. How things connect and read input and output and all the different ways of getting and handling in and output really confused me. I got better over the week in understanding, but the logic behind Sockets and BufferedReader etc. still sort of puzzle me at times. I think this lab was really fun, once you got the hang of it a little bit or understood other bits of code that we found online and what they meant, but it was quite a challenging lab.

Pavel: This exercise was a bit tough even with the help we found online, because in the beginning we weren't able to come up with many ideas. I learned more about socket programming and how the Server and Client communicate with each other. It was really necessary to go step by step with the bits of code, because it was something we haven't done before and it was confusing at some points. After we tried if different lines of code work or not, the process got easier and it was interesting to work on the exercise.

Appendix

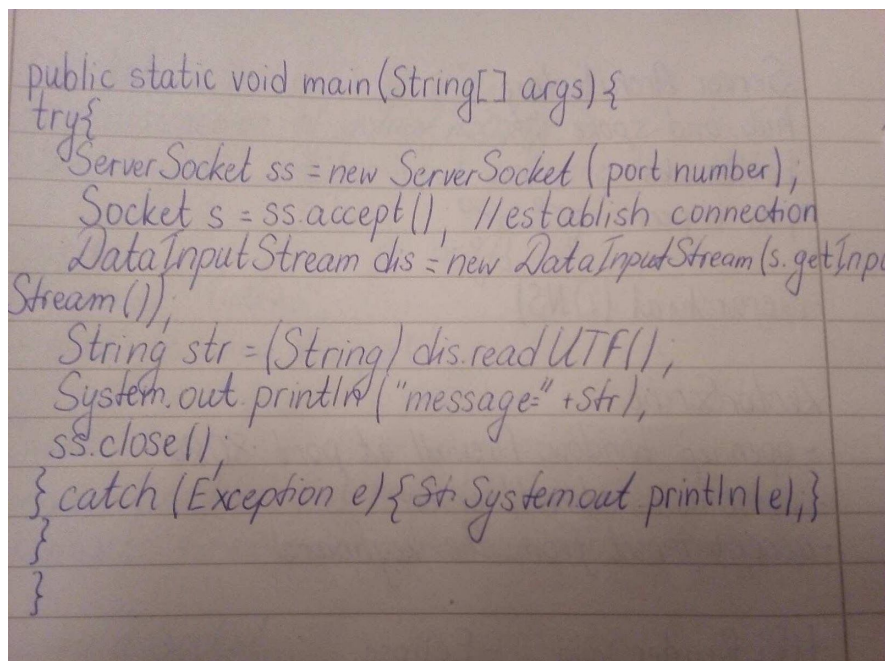
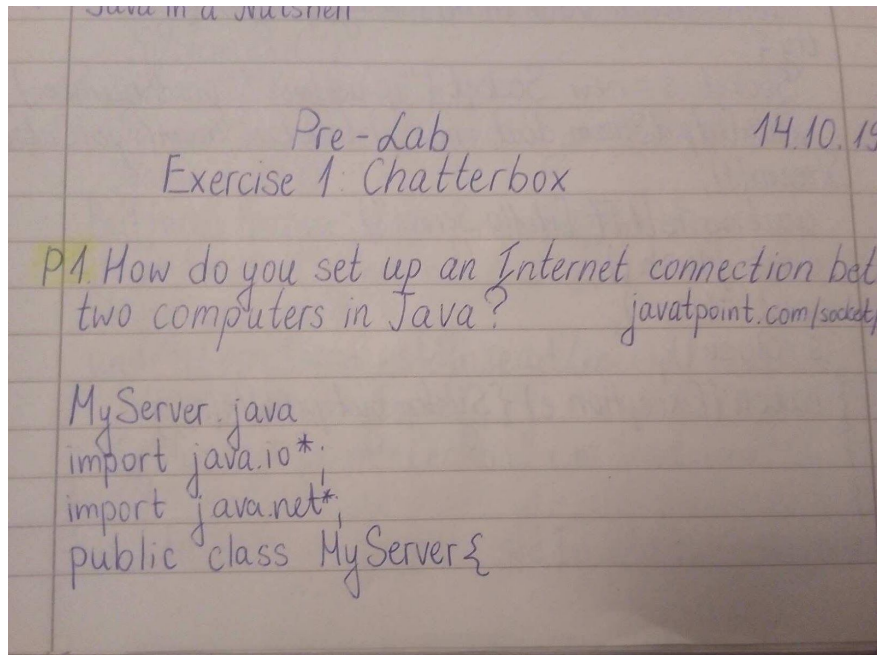
Sources

For exercise 1 and 2 we used help online from the following site:

<https://www.javatpoint.com/socket-programming>

Pre-Lab

Pavel:



math.uni-hamburg.de/doc/java/tutorial/networking/urls/readingWriting.html
docs.oracle.com/javase/tutorial/networking/urls/readingWriting.html

P2 Write a method to read from a Connection in Java
Reading from a URL Connection

```
import java.net.*;
import java.io.*;

public class (URL Connection Reader) EchoClient {
    public static void main (String[] args) throws IOException {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket ("ip address", port Number);
            out = new PrintWriter (echoSocket.getOutputStream(), true);
            in = new BufferedReader (new InputStreamReader (echoSocket
            .getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println ("Don't know about host: " + "ip address");
            System.exit (1);
        } catch (IOException e) {
            System.err.println ("Couldn't get I/O for " + "ip address");
            System.exit (1);
        }
        // Read
        {
            BufferedReader stdIn = new BufferedReader (new Input-
            Stream Reader (System.in));
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println (userInput);
                System.out.println ("echo: " + in.readLine());
            }
            out.close(); in.close(); stdIn.close(); echoSocket.close();
        }
    }
}
```

P3 Write a method to write to a connection in Java

```
String inputLine, outputLine;
KnockKnockProtocol kkp = new KnockKnockProtocol();
outputLine = kkp.processInput (null);
out.println (outputLine);
```

```
while ((inputLine = in.readLine()) != null) {
    outputLine = kkp.processInput (inputLine);
    out.println (outputLine);
    if (outputLine.equals ("Bye"))
        break;
}
```


Lena:

Pe-lab 1 14.10.19

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

```
import java.net.*;
import java.io.*;

public class Client {
    private Socket socket = null;
    private DataInputStream input = null;
    private DataOutputStream out = null;

    public Client (String address, int port) {
        try {
            socket = new Socket (address, port);
            System.out.println ("Connected");
            input = new DataInputStream (socket.getInputStream());
        }
        catch (UnknownHostException u) {
            System.out.println (u);
        }
        catch (IOException i) {
            System.out.println (i);
        }
    }
}
```

```
String line = ""; // String to read message from input
while (!line.equals ("Over")) {
    try {
        line = input.readLine();
        out.writeUTF (line);
    }
    catch (IOException i) {
        System.out.println (i);
    }
} // close the connection
try {
    input.close();
    out.close();
    socket.close();
}
catch (IOException i) {
    System.out.println (i);
}

public static void main (String[] args) {
    Client client = new Client (IP, Port);
}
```

```
import java.net.*;
import java.io.*;

public class Server {
    private Socket socket = null;
    private ServerSocket server = null;
    private DataInputStream in = null;

    public Server (int port) {
        try {
            server = new ServerSocket (port);
            System.out.println ("Server started");
            // ("Waiting for a client...");
            socket = server.accept();
            System.out.println ("Client accepted");
            in = new DataInputStream (new BufferedInputStream (socket.getInputStream()));
        }

        String line = "";
        while (!line.equals ("Over")) {
            try {
                line = in.readUTF();
                System.out.println (line);
            }
        }
    }
}
```

```

    catch (IOException i) {
        System.out.println(line);
    }
    System.out.println("Closing connection");
    socket.close();
    in.close();
}

catch (IOException i) {
    System.out.println(i);
}
}

public static void main (String[] args) {
    Server server = new Server(port);
}

```

www.geeksforgeeks.org/socket-programming-in-java/

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

```

public void connectionReader () {
    URL address = new URL ("url");
    URLConnection open = address.openConnection();
    Reader in = new Reader (new InputStreamReader
        (open.getInputStream()));

    String inputline;
    while ((inputline = in.readLine()) != null)
        System.out.println(inputline);
    in.close();
}

```

3. Reading from and Writing to a URLConnection, oracle

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

```

public void connectionWriter () {
    String userInput;
    while ((userInput = stdin.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
    }
}

```

4. Reading and Writing to a Socket, oracle

Code

Client Class

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;

class Client implements Runnable {
    Socket socket;

    public static void main(String[] args) {
        new Thread(new Client()).start();
    }

    @Override
    public void run() {
        try {
            socket = new Socket("localhost", 51200);
            BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
            DataOutputStream outputStream = new
DataOutputStream(socket.getOutputStream());

            System.out.println("Write a message...");

            while(true) {
                String inputString = reader.readLine();
                outputStream.writeUTF(inputString);
                outputStream.flush();

                if (inputString.equals("bye")) {

                    break;
                }
            }

            socket.close();
            reader.close();
            outputStream.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

```
}
```

Server Class

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
```

```
class Server implements Runnable {
    ServerSocket serverSocket;

    public static void main(String[] args) {
        new Thread(new Server()).start();
    }

    @Override
    public void run() {

        try {
            serverSocket = new ServerSocket(51200);
            Socket readerSocket = serverSocket.accept();
            DataInputStream inputReader = new DataInputStream(new
BufferedReaderInputStream(readerSocket.getInputStream()));

            while (true) {
                String messageString = inputReader.readUTF();

                if (messageString.equals("bye")) {
                    System.out.println("Server says: Bye!");
                    break;

                } else {
                    System.out.println("Client said: " + messageString);
                }
            }

            readerSocket.close();
            serverSocket.close();
            inputReader.close();

        } catch (IOException e) {

            System.out.println(e);
        }
    }
}
```

