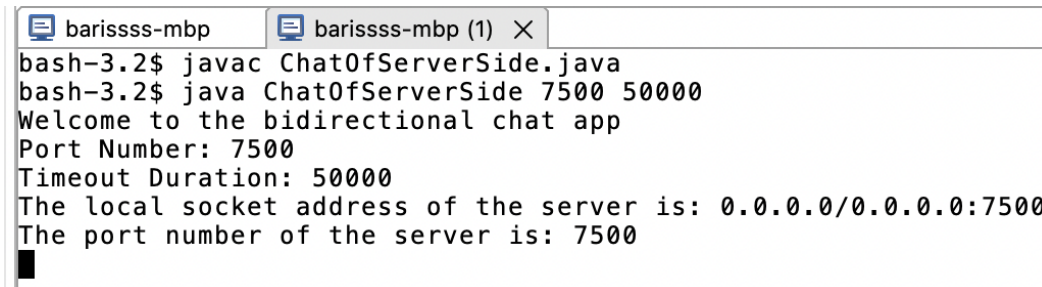


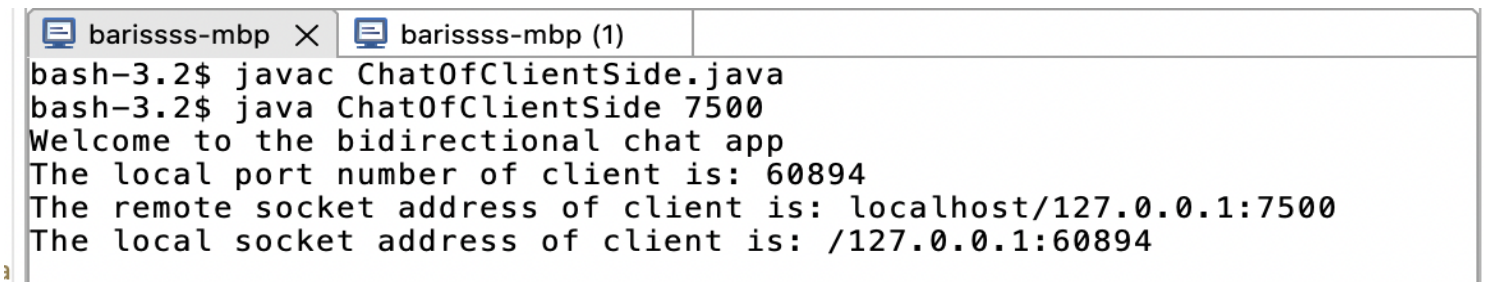
COMP416 PROJECT-1 PART-1 REPORT

Name-Surname: Barış Kaplan , KU ID Number: 0069054



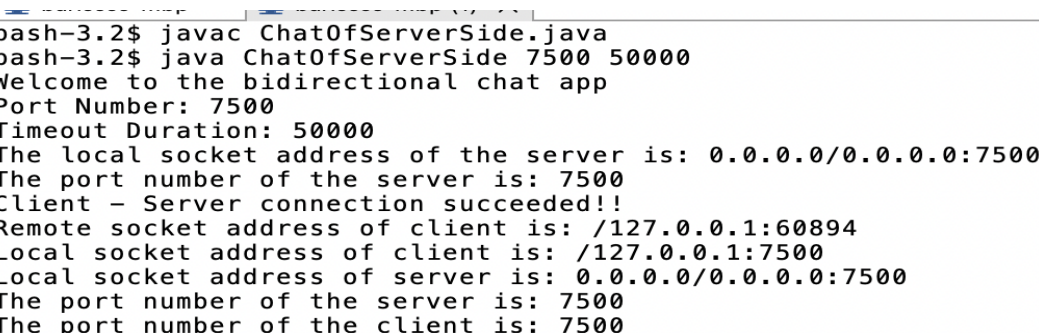
```
barissss-mbp  barissss-mbp (1) X
bash-3.2$ javac ChatOfServerSide.java
bash-3.2$ java ChatOfServerSide 7500 50000
Welcome to the bidirectional chat app
Port Number: 7500
Timeout Duration: 50000
The local socket address of the server is: 0.0.0.0/0.0.0.0:7500
The port number of the server is: 7500
```

Figure 1: The screenshot which shows the port number and the timeout duration given as arguments from the console. (This screenshot is taken from the server-side terminal). Here, the first argument is the port number (Port number = 7500). Moreover, the second argument is the timeout duration (Timeout duration = 50000 ms).



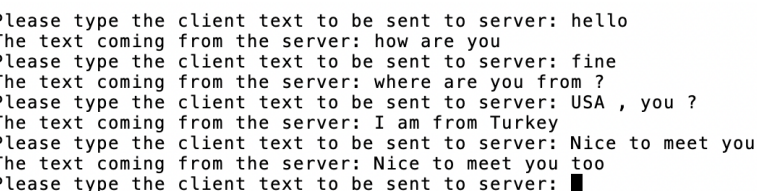
```
barissss-mbp X  barissss-mbp (1)
bash-3.2$ javac ChatOfClientSide.java
bash-3.2$ java ChatOfClientSide 7500
Welcome to the bidirectional chat app
The local port number of client is: 60894
The remote socket address of client is: localhost/127.0.0.1:7500
The local socket address of client is: /127.0.0.1:60894
```

Figure 2: The screenshot which shows the local port number of client, the remote socket address of the client, and the local socket address of the client (This screenshot is taken from the client-side terminal).



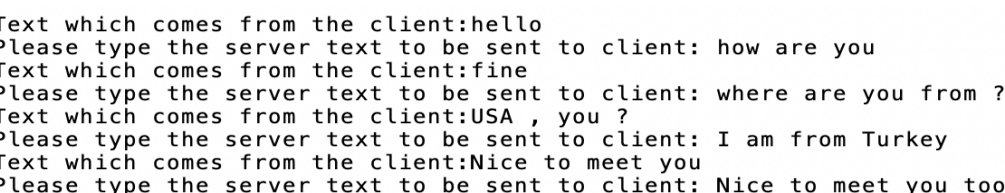
```
bash-3.2$ javac ChatOfServerSide.java
bash-3.2$ java ChatOfServerSide 7500 50000
Welcome to the bidirectional chat app
Port Number: 7500
Timeout Duration: 50000
The local socket address of the server is: 0.0.0.0/0.0.0.0:7500
The port number of the server is: 7500
Client - Server connection succeeded!!
Remote socket address of client is: /127.0.0.1:60894
Local socket address of client is: /127.0.0.1:7500
Local socket address of server is: 0.0.0.0/0.0.0.0:7500
The port number of the server is: 7500
The port number of the client is: 7500
```

Figure 3: The screenshot which shows the remote socket address of the client , the local socket address of client, the local socket address of server, the port number of server, and the port number of client (This screenshot is taken from the server-side terminal).



```
Please type the client text to be sent to server: hello
The text coming from the server: how are you
Please type the client text to be sent to server: fine
The text coming from the server: where are you from ?
Please type the client text to be sent to server: USA , you ?
The text coming from the server: I am from Turkey
Please type the client text to be sent to server: Nice to meet you
The text coming from the server: Nice to meet you too
Please type the client text to be sent to server: 
```

Figure 4: The screenshot which shows the chat in the client (This screenshot is taken from the client-side terminal).



```
Text which comes from the client:hello
Please type the server text to be sent to client: how are you
Text which comes from the client:fine
Please type the server text to be sent to client: where are you from ?
Text which comes from the client:USA , you ?
Please type the server text to be sent to client: I am from Turkey
Text which comes from the client:Nice to meet you
Please type the server text to be sent to client: Nice to meet you too
```

Figure 5: The screenshot which shows the chat in the server (This screenshot is taken from the server-side terminal).

Please type the client text to be sent to server: bye
bye is inputted by the client !
Chat App ended !
bash-3.2\$ █

Figure 6: The screenshot which shows the keyword I used to end the chat, and the messages displayed at the end of the chat (This screenshot is taken from the client-side terminal).

bye is inputted by the client !
Chat App ended !
bash-3.2\$

Figure 7: This screenshot is taken from the server-side terminal. Here; after client enters bye to terminal, server is informed from that. Then, at the end of the program, the messages in this screenshot are displayed in the server-side terminal

```
bash-3.2$ javac ChatOfServerSide.java
bash-3.2$ java ChatOfServerSide 7500 50000
Welcome to the bidirectional chat app
Port Number: 7500
Timeout Duration: 50000
The local socket address of the server is: 0.0.0.0/0.0.0.0:7500
The port number of the server is: 7500
No communication occurred between the server and the client !
timeout in socket !
java.net.SocketTimeoutException: Accept timed out
    at java.base/sun.nio.ch.NioSocketImpl.timedAccept(NioSocketImpl.java:705)
    at java.base/sun.nio.ch.NioSocketImpl.accept(NioSocketImpl.java:749)
    at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:673)
    at java.base/java.net.ServerSocket.platformImplAccept(ServerSocket.java:639)
    at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:615)
    at java.base/java.net.ServerSocket.implAccept(ServerSocket.java:572)
    at java.base/java.net.ServerSocket.accept(ServerSocket.java:530)
    at ChatOfServerSide.main(ChatOfServerSide.java:42)
bash-3.2$ █
```

Figure 8: The screenshot which shows the timeout in server and the messages displayed after timeout (This screenshot is taken from the server-side terminal). (In this case, there is no client which is connected to the server.)

```
public class ChatOfServerSide {
    private static BufferedReader brs = null;
    private static String txtOfCli = "";
    private static BufferedReader bri = null;
    private static PrintWriter p = null;
    private static String txtOfSer = "";
    private static ServerSocket ser = null;
    private static Socket cli = null;
    //The initialization of the sockets, buffered readers, strings, and print writers to be used in the

    //Main Method//
    public static void main(String[] args) throws IOException, TimeoutException {
        try {
            System.out.println("Welcome to the bidirectional chat app");

            //converting the first argument (which represents port number)
            //to integer by using the Integer.valueOf method, and passing this number to the inside
            //of ServerSocket() constructor call.
            ser = new ServerSocket(Integer.valueOf(args[0]));

            //Displaying port number and timeout duration to console for observation purposes.
            System.out.println("Port Number: "+Integer.valueOf(args[0]));
            System.out.println("Timeout Duration: "+Integer.valueOf(args[1]));

            //converting the second argument (which represents the timeout duration)
            //to integer by using the Integer.valueOf() method and passing this
            //number to the inside of the setSoTimeout function to set the timeout duration
            //of the server socket.
            ser.setSoTimeout(Integer.valueOf(args[1]));
            System.out.println("The local socket address of the server is: " + ser.getLocalSocketAddress() + "
            System.out.println("The port number of the server is: " + ser.getLocalPort() + "");
            cli = ser.accept(); //connection with client.
            System.out.println("Client - Server connection succeeded!!");
            System.out.println("Timeout duration of client is: "+cli.getSoTimeout());
            if(cli.getSoTimeout() == 0) {
                System.out.println("The timeout duration of the client is not set to a specific value!");
            }
        }
    }
}
```

Figure 9: This screenshot shows the timeout settings of server and client, the usage of setSoTimeout(int k) method, obtaining the timeout duration from the console as the second argument, and converting the second console argument to Integer. (The timeout for client socket can be set by calling setSoTimeout() method with the client socket object called “cli”). You can see Figure 1 for observing the timeout duration that I have given as an argument. (args[1] = Timeout Duration , Timeout Duration = 50000 ms).

Client - Server connection succeeded!!
Timeout duration of client is: 0
The timeout duration of the client is not set to a specific value!

The screenshot which shows the default timeout settings of the client. (screenshot taken from ChatOfServerSide.java)

Timeout duration of the created client socket: 0
The timeout duration of the client is not set to a specific value!

The screenshot which shows the default timeout settings of client (screenshot taken from ChatOfClientSide.java)

READ ME SECTION:

You should firstly open two separate terminals in the IDE you work (One for the client, and another for the server). Inside the client terminal, to compile the client side code, you should type the command called “javac ChatOfClientSide.java” to terminal. Inside the server side terminal, to compile the server side code, you should type “javac ChatOfServerSide.java” to terminal. Then, in order to run the server side code, I have typed “java ChatOfServerSide 7500 50000” to the terminal.

CONTINUATION OF THE READ ME SECTION:

In the “java ChatOfServerSide 7500 50000” command, I have given 7500 to the port number (which is the first argument (args[0]) in the command), and 50000 to the timeout duration (which is the second argument (args[1]) in the command). You can provide whichever value you want to the port number and to the timeout duration in the command for running the server side code. (this value should be in the suitable ranges for both of the arguments) Then, to make the client join to the server, I have typed the command “java ChatOfClientSide 7500” to the client side terminal. In this command, I have given the port number (7500) as the first argument. As long as the port number is in the suitable range, you can give whichever port number you want as an argument to the command for running the client side code.


```

1 import java.net.*;
2 import java.io.*;
3 import java.util.InputMismatchException;
4 import java.util.Objects;
5
6
7 public class ChatOfClientSide {
8     private static String hostType = "";
9     private static String zerothArgument = "";
10    private static int portNum = 0;
11    private static String textOfCli = "";
12    private static Socket cliSock = null;
13    private static PrintWriter pw = null;
14    private static BufferedReader bufReadStr = null;
15    private static String textOfSer = "";
16    private static BufferedReader bufReadInp = null;
17

```

Here, I have imported the “.net”, “.io”, “.util.InputMismatchException”, and “.Objects” libraries. After that, I have initialized the necessary variables to be used in the client side code.

```

195
196 public static void displaySocketAddressAndPortInfo() {
197     int localPortNum = cliSock.getLocalPort(); //obtaining the port of client socket by usin
198
199     SocketAddress remoteCli = cliSock.getRemoteSocketAddress(); //obtaining the remote sock
200     SocketAddress localCli = cliSock.getLocalSocketAddress(); //obtaining the local socket a
201
202
203     System.out.println("The local port number of client is: " + localPortNum);
204     System.out.println("The remote socket address of client is: " + remoteCli + "");
205     System.out.println("The local socket address of client is: " + localCli + "");
206 }
207
208 ///////////////////////////////////////////////////Helper functions////////////////////////////////////
209

```

Inside the above method; by using getLocalPort() method, I have obtained the local port number of the client socket. By using getRemoteSocketAddress() method, I have obtained the remotesocket address of client. By using getLocalSocketAddress() method, I have obtained the local socket address of client.

```

110
111 public static void checkClientAndServerTexts() {
112     while (true) {
113         System.out.print("Please type the client text to be sent to server: ");
114         try {
115             textOfCli = bufReadInp.readLine(); //obtaining the client input
116         } catch (IOException e) {
117             // TODO Auto-generated catch block
118             e.printStackTrace();
119         }
120
121         boolean isInSwitchAndTextBye = false; //checks whether the code is inside the switch and
122         boolean loopEnds = false; //creating a boolean variable called loopEnds and assigning it
123
124
125
126
127         pw.println(textOfCli);
128
129         switch(textOfCli) {
130
131             case "": //if the client input is blank string, then assign the boolean variable call
132                 System.out.println("Blank is typed by the client !");
133                 System.out.println("Chat App terminated !");
134                 loopEnds = true;
135                 break;
136             case "bye": //if the client input is "bye", then assign the boolean variable called l
137                 isInSwitchAndTextBye = true;
138                 System.out.println("Bye is inputted by the client !");
139                 System.out.println("Chat App ended !");
140                 loopEnds = true; //when the client input is equal to "bye", assign loopEnds variab
141                 break;
142             default:
143                 break;
144         }
145
146         if(textOfCli.equalsIgnoreCase("bye") && isInSwitchAndTextBye == false) { //check whether
147             System.out.println("Bye is inputted by the client !");
148             System.out.println("Chat App ended !");
149             loopEnds = true; //assign loopEnds variable to true
150         }
151
152         if(loopEnds == true) {
153             break;
154         }
155
156         try {
157             textOfSer = bufReadStr.readLine(); //obtaining the server input
158         } catch (IOException e) {
159             // TODO Auto-generated catch block
160             e.printStackTrace();
161         }
162
163         switch(textOfSer) {
164             case "": //checking whether the server input is equal to the blank string.
165                 System.out.println("Blank is typed by the server !");
166                 System.out.println("Chat App terminated !");
167                 loopEnds = true;
168                 break;
169             case "bye": //checking whether the server input is equal to "bye"
170                 isInSwitchAndTextBye = true;
171                 System.out.println("Bye is inputted by the server !");
172                 System.out.println("Chat App ended !");
173                 loopEnds = true;
174                 break;
175             default:
176                 break;
177         }
178
179         if(loopEnds == true) { //if the value of loopEnds is true
180             break; //terminate the loop
181         }
182
183         System.out.println("The text coming from the server: " + textOfSer + "");
184     }
185 }

```

Check Client And Server Texts SS1

Then, inside the client side code, I have created a method for obtaining the client inputs and keeping track of client inputs & server inputs (See Check Client And Server Texts SS1 and. Check Client And Server Texts SS2). I have obtained the client inputs by using readLine() method. By using separate switch cases for client texts and server texts, I have checked whether the client input or server input is equal to bye or blank. For terminating the while loop when the input is bye or blank, I have defined a boolean variable called “loopEnds”. I have made this variable true for the client inputs and server inputs which are equal to “bye” or blank. By using the equalsIgnoreCase() method, I have also checked the cases of the client inputs and server inputs equal to “bye”. In these checks, I have made the “loopEnds” variable true. By using println function with the print writer object (pw.println(text)), I have attached the input of the client to the print writer object. In the server side, I have defined a method which is functionally similar to the method in the above screenshot. Whereas I have taken client inputs in the client side code, I have taken the server side inputs in the server side code.

```

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

        System.out.println("Welcome to the bidirectional chat app");
        boolean flushable = true;

        zerothArgument = args[0];
        portNum = Integer.valueOf(zerothArgument);
        hostType = "localhost";

        cliSock = new Socket(hostType, portNum);
        OutputStream clieOut = cliSock.getOutputStream();
        pw = new PrintWriter(clieOut, flushable);

        InputStream clieIn = cliSock.getInputStream();
        InputStreamReader clieInRead = new InputStreamReader(clieIn);
        bufReadStr = new BufferedReader(clieInRead);

```

```

        InputStreamReader terminalInput = new InputStreamReader(System.in);
        bufReadInp = new BufferedReader(terminalInput);

```

Next, I have taken the port number as an argument. For that purpose, I have used the Integer.valueOf() method. I have then created the client socket, and the streams & stream readers which will be used to keep track of the client inputs and server inputs.

```

displaySocketAddressAndPortInfo();
System.out.println();
checkClientAndServerTexts();

```

After I created the streams, stream readers, and the client socket; I have called the methods called displaySocketAddressAndPortInfo() and checkClientAndServerTexts() in the main method of the client side code.

```

switch(textOfSer) {
    case "": //checking whether the server input is equal to the blank string.
        System.out.println("Blank is typed by the server !");
        System.out.println("Chat App terminated !");
        loopEnds = true; //if the server input is blank string, assign the loopEnds to true
        break;
    case "bye": //checking whether the server input is equal to "bye"
        isInSwitchAndTextBye = true;
        System.out.println("Bye is inputted by the server !");
        System.out.println("Chat App ended !");
        loopEnds = true; //if the server input is bye, then assign the loopEnds variable
        break;
    default:
        break;
}

if(textOfSer.equalsIgnoreCase("bye") && isInSwitchAndTextBye == false) { //check whether
    System.out.println("Bye is inputted by the server !");
    System.out.println("Chat App ended !");
    loopEnds = true;
}

if(loopEnds == true) { //if the value of loopEnds is true
    break; //terminate the loop
}

System.out.println("The text coming from the server: " + textOfSer + "");
}

```

Check Client And Server Texts SS2

```

} catch (NullPointerException npexc) {
    System.out.println("Null pointer error has happened !");
    npexc.printStackTrace();
} catch (SocketTimeoutException ste) {
    System.out.println("No activity in the server !");
    System.out.println("timeout in socket !");
    ste.printStackTrace();
} catch (InputMismatchException ime) {
    System.out.println("You have entered an input which is not a text");
    ime.printStackTrace();
} catch (SocketException sockEx) {
    System.out.println("Socket error happened !");
    sockEx.printStackTrace();
} catch (ArithmeticException aexc) {
    System.out.println("Arithmetic error happened !");
    aexc.printStackTrace();
} catch (IOException ioEx) {
    System.out.println("IO Exception has happened !");
    ioEx.printStackTrace();
}

//creating a boolean variable which checks of whether the sockets,
//buffered readers, and print writers are null.
boolean objectsNotNull = false;

objectsNotNull =
    (Objects.isNull(pw) == false) &&
    (Objects.isNull(bufReadStr) == false) &&
    (Objects.isNull(bufReadInp) == false) &&
    (Objects.isNull(cliSock) == false);

//if all objects are not null
if(objectsNotNull == true) {
    bufReadStr.close(); //close the buffered reader object which keeps track of incomin
    pw.close(); //close the print writer object
    bufReadInp.close(); //close the buffered reader object which keeps track of the in
    cliSock.close(); //close the client socket
}
}

```

Exception Handling and Object Closing SS

```

2 import java.util.InputMismatchException;
3 import java.util.Objects;
4 import java.util.concurrent.TimeoutException;
5 import java.io.*;
6
7 public class ChatOfServerSide {
8     private static BufferedReader brs = null;
9     private static String txtOfCli = "";
10    private static BufferedReader bri = null;
11    private static PrintWriter p = null;
12    private static String txtOfSer = "";
13    private static ServerSocket ser = null;
14    private static Socket cli = null;
15
16    public static void main(String[] args) throws IOException, TimeoutException {
17        try {
18
19            System.out.println("Welcome to the bidirectional chat app");
20            ser = new ServerSocket(Integer.valueOf(args[0]));
21            System.out.println(Integer.valueOf(args[1]));
22            ser.setSoTimeout(Integer.valueOf(args[1]));
23            System.out.println("The local socket address of the server is: " + ser.getLocalSocketAddress());
24            System.out.println("The port number of the server is: " + ser.getLocalPort() + "");
25            cli = ser.accept();
26            System.out.println("Client - Server connection succeeded!!");
27            showSocketAddressAndPortInfo();
28            createStreams();
29            System.out.println();
30            checkInputsOfClientAndServer();
31
32        } catch (SocketTimeoutException ste) {
33            System.out.println("No communication occurred between the server and the client !");
34            System.out.println("timeout in socket !");
35            ste.printStackTrace();
36        } catch (SocketException e) {
37            System.out.println("Socket error happened !");
38            e.printStackTrace();
39        } catch (NullPointerException npexc) {
40            System.out.println("Null pointer error has happened !");
41            npexc.printStackTrace();
42        } catch (InputMismatchException ime) {
43            System.out.println("You have entered an input which is not a text");
44            ime.printStackTrace();
45        } catch (ArithmeticException aexc) {
46            System.out.println("Arithmetic error happened !");
47            aexc.printStackTrace();
48        } catch (IOException e) {
49            System.out.println("IO exception has happened !");
50            e.printStackTrace();
51        }
52    }

```

Server Side Code SS1

```

154 public static void checkInputsOfClientAndServer() {
155     while(true) {
156         try {
157             txtOfCli = brs.readLine(); //reading the client input
158         } catch (IOException e) {
159             // TODO Auto-generated catch block
160             e.printStackTrace();
161         }
162
163         boolean inSwitchAndTextBye = false; //checks whether the code is in switch and text is b
164         boolean terminate = false; //creating a boolean variable which checks the termination of
165
166         switch(txtOfCli) {
167             case "": //checking whether the client input is blank.
168                 System.out.println("Blank is entered by the client !");
169                 System.out.println("Chat App terminated !");
170                 terminate = true; //if the client input is blank string, make the boolean variabl
171                 break;
172             case "bye": //checking whether the client is input is bye.
173                 System.out.println("Bye is inputted by the client !");
174                 terminate = true; //if the client input is "bye", make the boolean variable call
175                 inSwitchAndTextBye = true;
176                 break;
177             default:
178                 break;
179         }
180
181         if(txtOfCli.equalsIgnoreCase("bye") && inSwitchAndTextBye == false) { //checking whether
182             System.out.println("Bye is inputted by the client !");
183             System.out.println("Chat App ended !");
184             terminate = true;
185         }
186
187         if(terminate == true) { //if the value of the boolean variable called terminate is true
188             break; //terminate the while loop
189         }
190
191         System.out.println("Text which comes from the client:" + txtOfCli + " ");
192         System.out.println("Please type the server text to be sent to client:");
193         try {
194             txtOfSer = bri.readLine(); //reading the server input
195         } catch (IOException e) {
196             // TODO Auto-generated catch block
197             e.printStackTrace();
198         }
199         p.println(txtOfSer); //attaching the server input
200
201         switch(txtOfSer) {
202             case "": //checking whether the server input is equal to blank.
203                 System.out.println("Blank is entered by the server !");
204                 System.out.println("Chat App terminated !");
205                 terminate = true; //if the server input is blank string, make the boolean variab
206                 break;
207             case "bye": //checking whether the server input is equal to bye.
208                 System.out.println("Bye is typed by the server !");
209                 System.out.println("Chat App terminated !");
210                 terminate = true; //if the server input is "bye", make the boolean variable call
211                 inSwitchAndTextBye = true;

```

Server Side Code SS3

Here, inside the client side code (see Exception Handling and Object Closing SS) , I have handled some possible exceptions. Then, I have closed all buffered reader, print writer, and socket objects. While closing these objects, in order to deal with the timeout situation, I have checked whether these objects are not null by using the isNull method of the Objects class of Java. In the server-side code (see Server Side Code SS2), I have followed the same approach to handle the exceptions and close the objects.

I have handled the exceptions in the catch blocks.

For checking whether the buffered reader objects, print writer objects, and socket objects are null, I have used Objects.isNull() method. If all of the buffered reader objects, print writer objects, and socket objects are not null, then I have closed these objects by using close() method.

Server Side SS1: I have set the timeout of the server by using setSoTimeout() method. I have dealt with exceptions in separate catch blocks. For converting the command line arguments to integers, I have utilized Integer.valueOf() method.

Server Side SS2: Inside the method called "showSocketAddressAndPortInfo()", by using the getRemoteSocketAddress() method, I have displayed the remote socket address of the client socket. By using getLocalSocketAddress() method with client socket and server socket, I have displayed the local socket addresses of them.

```

68 //create a boolean variable which keeps track of whether the sockets,
69 //buffered readers, and print writers are null.
70 boolean areNotNull = false;
71
72 areNotNull =
73     Objects.isNull(cli) == false &&
74     Objects.isNull(ser) == false &&
75     Objects.isNull(brs) == false &&
76     Objects.isNull(p) == false &&
77     Objects.isNull(bri) == false;
78
79
80 // if all of the objects are not null, close all of the objects.
81 if(areNotNull == true) {
82     cli.close();
83     ser.close();
84     p.close();
85     brs.close();
86     bri.close();
87 }
88
89
90 }
91
92
93
94 public static void showSocketAddressAndPortInfo() {
95     //getting the remote socket address of client by using getRemoteSocketAddress() method
96     SocketAddress remoteClientAddr = cli.getRemoteSocketAddress();
97
98     //getting the local socket address of client by using getLocalSocketAddress() method
99     SocketAddress localClientAddr = cli.getLocalSocketAddress();
100
101     //getting the local socket address of server by using getLocalSocketAddress() method
102     SocketAddress localServerAddr = ser.getLocalSocketAddress();
103
104     int portOfServer = ser.getLocalPort(); //obtaining the local port of server by using
105     int portOfClient = cli.getLocalPort(); //obtaining the local port of client by using
106
107     //displaying the remote socket address of client by utilizing getRemoteSocketAddress()
108     System.out.println("Remote socket address of client is: " + remoteClientAddr + "");
109
110     //displaying the local socket address of client by utilizing getLocalSocketAddress() me
111     System.out.println("Local socket address of client is: " + localClientAddr + "");
112
113     //displaying the local socket address of server by utilizing getLocalSocketAddress() me
114     System.out.println("Local socket address of server is: " + localServerAddr + "");
115
116     //Displaying the port number of server and client by using getLocalPort() method.
117     System.out.println("The port number of the server is: " + portOfServer + "");
118     System.out.println("The port number of the client is: " + portOfClient + "");
119
120 }
121
122 public static void createStreams() {
123     InputStream cliIn = null;
124     boolean isFlushable = true;
125     try {

```

Server Side Code SS2

```

111 cliIn = cli.getInputStream();
112 } catch (IOException e) {
113     // TODO Auto-generated catch block
114     e.printStackTrace();
115 }
116
117 InputStreamReader cliInRead = new InputStreamReader(cliIn);
118 brs = new BufferedReader(cliInRead); //creating a buffered reader object which is for
119
120 OutputStream cliOut = null;
121 try {
122     cliOut = cli.getOutputStream();
123 } catch (IOException e) {
124     // TODO Auto-generated catch block
125     e.printStackTrace();
126 }
127
128 p = new PrintWriter(cliOut, isFlushable); //for writing to client output
129
130 InputStreamReader consoleInp = new InputStreamReader(System.in);
131 bri = new BufferedReader(consoleInp);

```

Continuation of createStreams() function

In the server-side code, I have taken the port number and timeout duration as arguments. I have taken the port number as the first argument and the timeout duration as the second argument. Since these arguments are both String, I have converted them to integer by using Integer.valueOf() function. After that, I have displayed the local socket address (by using getLocalSocketAddress() function) and port (by using getLocalPort() function) info of the server. Next, by using accept() function, I have made the connection between the client and server. Then; I have created a method for displaying the remote socket address & local socket address of client, local socket address of server, the port number of server, and the port number of client. I have used getRemoteSocketAddress() method for displaying the remote socket address of the client, getLocalSocketAddress() method for displaying local socket addresses, and getLocalPort() for displaying port numbers. Then, inside a method called "createStreams()", I have created the streams which are used for keeping track of the client inputs and server inputs. Then; inside another method, I have obtained the client inputs with readLine() method. After that, I have checked whether the client inputs or server inputs is equal to "bye" or blank. I have done these checks in two separate switch statements. By using equalsIgnoreCase() method, I have also checked whether the client input or server input is equal to "bye" regardless of the case of "bye". I have defined a boolean variable which checks whether the input is "bye" and the code enters to the switch statement. Finally, inside the main method, I have closed all socket objects, print writer objects, and buffered reader objects. In order to handle timeout situation, while closing these objects, I have checked whether these objects are null by using Objects.isNull() method.

```
java.net.SocketTimeoutException: Read timed out
    at java.base/sun.nio.ch.NioSocketImpl.timedRead(NioSocketImpl.java:280)
    at java.base/sun.nio.ch.NioSocketImpl.implRead(NioSocketImpl.java:306)
    at java.base/sun.nio.ch.NioSocketImpl.read(NioSocketImpl.java:347)
    at java.base/sun.nio.ch.NioSocketImpl$1.read(NioSocketImpl.java:800)
    at java.base/java.net.Socket$SocketInputStream.read(Socket.java:966)
    at java.base/sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:270)
    at java.base/sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:313)
    at java.base/sun.nio.cs.StreamDecoder.read(StreamDecoder.java:188)
    at java.base/java.io.InputStreamReader.read(InputStreamReader.java:176)
    at java.base/java.io.BufferedReader.fill(BufferedReader.java:162)
    at java.base/java.io.BufferedReader.readLine(BufferedReader.java:329)
    at java.base/java.io.BufferedReader.readLine(BufferedReader.java:396)
    at ChatOfServerSide.checkInputsOfClientAndServer(ChatOfServerSide.java:168)
    at ChatOfServerSide.main(ChatOfServerSide.java:55)
```

The figure above is the read timeout output from the server side terminal. (If we specify the timeout value of the client after accepting the client in the server side code). For specifying the timeout duration of client, cli.setSoTimeout(Integer.valueOf(args[1])) method should be used. Since the type of an argument which comes from the command line is String, we should convert it to integer by using the method called Integer.valueOf(args[1]). For making timeout duration equal to second command line argument, we get the timeout duration from the command line input by using args[1]. After the read timeout output in the server side terminal, the program will continue to want an input text from the client side. (This above screenshot is taken from the output (output of the run of source code called ChatOfServerSide.java) in the server side terminal.). For the above screenshot, client is connected to server but client does not provide any input text.

```
bash-3.2$ javac ChatOfServerSide.java
bash-3.2$ java ChatOfServerSide 7500 50000
Welcome to the bidirectional chat app
Port Number: 7500
Timeout Duration : 50000
The local socket address of the server is: 0.0.0.0/0.0.0.0:7500
The port number of the server is: 7500
Client - Server connection succeeded!!
Timeout duration of client is: 50000
```

Screenshot which shows the value of the client timeout duration after its specification with cli.setSoTimeout method.

(This screenshot shows the non-default timeout settings of client)

NOTE1: Timeout duration : 50000 (refers to server's timeout duration)

NOTE2: Timeout duration of client is: 50000 (refers to client's timeout duration)

```
//of ServerSocket() constructor call.
ser = new ServerSocket(Integer.valueOf(args[0]));

//Displaying port number and timeout duration to console for observat
System.out.println("Port Number: "+Integer.valueOf(args[0]));
System.out.println("Timeout Duration : "+Integer.valueOf(args[1]));

//converting the second argument (which represents the timeout durat
//to integer by using the Integer.valueOf() method and passing this
//number to the inside of the setSoTimeout function to set the timeo
//of the server socket.
ser.setSoTimeout(Integer.valueOf(args[1]));
System.out.println("The local socket address of the server is: " + .
System.out.println("The port number of the server is: " + ser.getLoc
cli = ser.accept(); //connection with client.
System.out.println("Client - Server connection succeeded!!");

//Please Read Here//
//NOTE: I have put both the default and non-default timeout setting:

//Please Read Here//
//for specifying the timeout duration of the client, the below comm
cli.setSoTimeout(Integer.valueOf(args[1])); //you can comment out tl
```

Screenshot which shows the how to specify the value of client timeout duration (Please see cli.setSoTimeout(Integer.valueOf(args[1]))

```
Text which comes from the client:hi
Please type the server text to be sent to client: bye
bye is typed by the server !
Chat App terminated !
bash-3.2$ █
```

Screenshot which shows the keyword I used for ending the program and the messages displayed in the server-side terminal at the end of the program (This screenshot is taken from the server-side terminal).

```
Please type the client text to be sent to server: hi
bye is inputted by the server !
Chat App ended !
bash-3.2$ █
```

This screenshot is taken from the client-side terminal. After server types bye to the terminal, client is informed from that. Moreover, the messages which are shown in this screenshot are printed to the client-side terminal.

SCREENSHOTS OF THE COMPILATION AND RUNNING OF CLIENT SIDE CODE AND SERVER SIDE CODE

```
bash-3.2$ javac ChatOfClientSide.java
bash-3.2$ java ChatOfClientSide 7500
```

Screenshot which shows the commands to compile and run the source code called ChatOfClientSide.java.

(This screenshot is taken from the client-side terminal).

```
bash-3.2$ javac ChatOfServerSide.java
bash-3.2$ java ChatOfServerSide 7500 50000
```

Screenshot which shows the commands to compile and run the source code called ChatOfServerSide.java

(This screenshot is taken from the server-side terminal).

BONUS PART:

To make the client and server run on different machines, inside the client-side code (namely ChatOfClientSide.java), we should initially obtain the host address of the ip address of server. For obtaining the host address of the ip address of server, we should use the `InetAddress.getLocalHost().getHostAddress()` method. This host address can also be obtained by typing the command called “`ipconfig getifaddr en0`” command to terminal in MacOS operating system. Subsequently; inside the client-side code, as the initial argument, we should pass the host address to the Socket constructor call of the client socket.

Instead of the process in the figure named “Without Bonus SS”, we can perform this process like in the figure named “Bonus SS” (below figure).

For compiling and running the client-side and server-side codes, we can follow the same approach that I have explained in the Read Me section of this report.

```
String hstAdr = InetAddress.getLocalHost().getHostAddress();
zerothArgument = args[0]; //obtaining port number as an argument in the command.
portNum = Integer.valueOf(zerothArgument); //converting the argument to integer by using Integer.valueOf()
cliSock = new Socket(hstAdr, portNum); //creating the client socket
```

Bonus SS (This screenshot is taken from the client-side code (Namely ChatOfClientSide.java)).

```
zerothArgument = args[0]; //obtaining port number as an argument in the command.
portNum = Integer.valueOf(zerothArgument); //converting the argument to integer by using Integer.valueOf()
hostType = "localhost"; //specifying the host
cliSock = new Socket(hostType, portNum); //creating the client socket
```

(Without Bonus SS), Creating Client Socket with localhost host type (This screenshot is taken from the client-side code (Namely ChatOfClientSide.java)).

```
[(base) barissss-mbp:~ barissss$ ipconfig getifaddr en0
192.168.1.5
(base) barissss-mbp:~ barissss$
```

The output of the run of "ipconfig getifaddr en0" command in macos terminal.

NOTE: Please have a look at the places which are marked with “Please Read Here” in the code.