Comp416 Project 1.1

## ReadMe

Firstly, open up a terminal window and change the directory to the project's directory by cd command. Compile the Client.java and Server.java files using "javac Server.java" and "javac Client.java" commands. Then execute the compiled files with necessary arguments using "java Server <port-number> <timeout-value>" and "java Client <port-number>". An example is given below.

```
burak@BURAK-MacBook-Pro ~ % cd /Users/burak/Desktop/KOC\ UNI/SEMESTERS/FALL\ 202
2/COMP416/src
burak@BURAK-MacBook-Pro src % javac Client.java
burak@BURAK-MacBook-Pro src % javac Server.java
```

## Execution

```
burak@BURAK-MacBook-Pro src % java Server 1111 35000
listening to port 1111
Connection established. The socket address: /127.0.0.1:58524
Timeout setting is: 35000
Client's message: Hi. I am the Client side.
Type in some text please. Type quit to terminate
Hi. I am the server side.
Client's message: What's up?
Type in some text please. Type quit to terminate
I am fine. What about you?
Client's message: I'm good. It's nice to hear from you again.
Type in some text please. Type quit to terminate
I gotta go. I hope to see you again soon.
Client's message: See you.
Type in some text please. Type quit to terminate
quit
burak@BURAK-MacBook-Pro src %
```

```
burak@BURAK-MacBook-Pro src % java Client 1111
Binding to port localhost:1111
Connection established. The socket address: localhost/127.0.0.1:1111
Timeout settings is: 35000
Type in some text please. Type quit to terminate
Hi. I am the Client side.
Server's message: Hi. I am the server side.
Type in some text please. Type quit to terminate
What's up?
Server's message: I am fine. What about you?
Type in some text please. Type quit to terminate
I'm good. It's nice to hear from you again.
Server's message: I gotta go. I hope to see you again soon.
Type in some text please. Type quit to terminate
See you.
Server's message: quit
burak@BURAK-MacBook-Pro src %
```

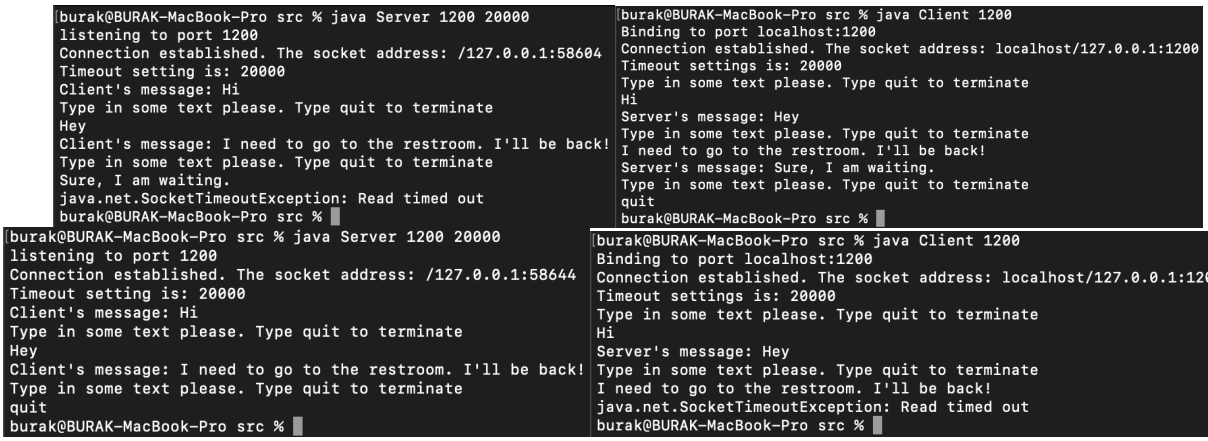
In the upper screenshots, left hand side is the server side and the right hand side is the client side.

1. Server side code is executed with port number as 1111 and timeout given as 35000 which refers to 35 seconds.
2. After establishing the connection, both sides show the socket addresses.
3. The timeout setting of the connected socket is shown for both sides.
4. An example chat between server and the client can be seen in the screenshots.

```
burak@BURAK-MacBook-Pro src % java Server 1500 40000
listening to port 1500
Connection established. The socket address: /127.0.0.1:58579
Timeout setting is: 40000
Client's message: Hi. I am the Client side.
Type in some text please. Type quit to terminate
Hi. I am the Server side.
Client's message: What's up?
Type in some text please. Type quit to terminate
I am fine. How are you?
Client's message: I am excellent!
Type in some text please. Type quit to terminate
I need to do my homework. See you later!
Client's message: quit
burak@BURAK-MacBook-Pro src %
```

```
burak@BURAK-MacBook-Pro src % java Client 1500
Binding to port localhost:1500
Connection established. The socket address: localhost/127.0.0.1:1500
Timeout settings is: 40000
Type in some text please. Type quit to terminate
Hi. I am the Client side.
Server's message: Hi. I am the Server side.
Type in some text please. Type quit to terminate
What's up?
Server's message: I am fine. How are you?
Type in some text please. Type quit to terminate
I am excellent!
Server's message: I need to do my homework. See you later!
Type in some text please. Type quit to terminate
quit
burak@BURAK-MacBook-Pro src %
```

5. When client or server side writes "quit" keyword chat is ended and connections are closed.




6. In case of a timeout like in the above screenshots, chat is ended and server closes the connection.

```java
import java.io.*;
import java.net.*;
public class Client {
    public static void main(String[] args) throws IOException {
        String host = "localhost";
        //Getting the port value as argument
        int port = Integer.parseInt(args[0]);
        int timeoutVal;
        Socket socket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader reader = null;
        System.out.println("Binding to port localhost:" + port);
        try {
            socket = new Socket ( host, port );
            out = new PrintWriter ( socket.getOutputStream (), true );
            in =
                    new BufferedReader (
                            new InputStreamReader ( socket.getInputStream () ) );
            reader =
                    new BufferedReader (
                            new InputStreamReader ( System.in ) );
            System.out.println("Connection established. The socket address: "+socket.getRemoteSocketAddress());
            String userInput;
            String inputLine;
            //Get timeout value from the server
            timeoutVal = Integer.parseInt(in.readLine());
            socket.setSoTimeout(timeoutVal);
            System.out.println("Timeout settings is: "+socket.getSoTimeout());
            System.out.println("Type in some text please. Type quit to terminate");
            while(true) {
                //Gets the input of client
                userInput = reader.readLine();
                //Sends the input to server
                out.println ( userInput );
                if(userInput.equals("quit")) {
                    break;
                }
                //Gets the message of server
                inputLine = in.readLine();
                System.out.println ("Server's message: " + inputLine);
                if(inputLine.equals("quit")) {
                    break;
                }
                System.out.println("Type in some text please. Type quit to terminate");
            }
        } catch (Exception e) {
            //Handling the socket timeout exception
            System.out.println(e);
            socket.close();
            out.close();
            in.close();
            reader.close();
        }
        //Close the sockets, readers and writers
        socket.close();
        out.close();
        in.close();
        reader.close();
    }
}
```

In the Client file, I get the port value from the user as an argument. Then, I initialized the socket, readers, and writers for the client-server interaction. After establishing the connection, I printed out the remote socket address and the timeout value. I initialized userInput and inputLine variables for getting the output of the client and receiving the message of the server respectively. Firstly, timeout value that is sent from the server is read and the timeout value is set to that specific value. Infinite loop that can only interrupted by exceeding the timeout limit or a quit message of either client or server side. In the loop, client and server writes a messages one by one. Client writes a message then it sends to the server. After that, the message of the server is received by the client. The procedure continuous until an interruption that I mentioned above. In case of an interruption, I inform the user about a quit operation or a timeout operation. In both cases, I close all the sockets, readers, and writers in the end.

```java
import java.net.*;
import java.io.*;
public class Server {
    public static void main(String[] args) throws IOException {
        //Get the port and timeout values as argument
        int port = Integer.parseInt(args[0]);
        int timeoutVal = Integer.parseInt(args[1]);
        System.out.println("listening to port " + port);
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader reader = null;
    try {
        serverSocket = new ServerSocket(port);
        clientSocket = serverSocket.accept();
        out =
                new PrintWriter(clientSocket.getOutputStream(), true);
        in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
        reader =
                new BufferedReader(
                        new InputStreamReader(System.in));
        clientSocket.setSoTimeout(timeoutVal);
        //Send timeout value to client
        out.println(timeoutVal);
        System.out.println("Connection established. The socket address: "+clientSocket.getRemoteSocketAddress());
        System.out.println("Timeout setting is: "+clientSocket.getSoTimeout());
        String inputLine;
        String serverInput;
        while(true) {
            //Gets the message of client
            inputLine = in.readLine();
            System.out.println ( "Client's message: " + inputLine );
            if(inputLine.equals("quit")) {
                break;
            }
            System.out.println("Type in some text please. Type quit to terminate");
            //Gets the input from server
            serverInput = reader.readLine();
            //Sends the input to client
            out.println(serverInput);
            if(serverInput.equals("quit")) {
                break;
            }
        }
        }catch (Exception e){
            //Handling the socket timeout exception
            System.out.println(e);
            serverSocket.close();
            clientSocket.close();
            out.close();
            reader.close();
            in.close();
        }
        //Close the sockets, readers and writers
        serverSocket.close();
        clientSocket.close();
        out.close();
        reader.close();
        in.close();

    }
}
```

In the server side, it is very similar to the client side. The port number and the timeout value are given as first and second argument from the user respectively. Server socket, client socket, writers, and readers are initialized. Timeout value is set for the socket and sent to the client. After establishing the connection, remote socket address and timeout values are printed out. Then, I defined inputLine and serverInput for getting the message of the client and giving the input of the server respectively. It waits for the client to send a message. After receiving the message of the client, the server is prompted to write a message. In the infinite loop, it continuous until an interruption. An interruption can only occur due to exceeding the timeout limit or a quit message of either client or server side. In case of a timeout exception it is handled by informing by printing out the exception. In both cases all the socket connections, readers, and writers are closed in the end.