Al-Imam University



**Computer Science Department**

**College of Computer and Information Sciences**

**CS330: Computer Networks**

**Course Project**

**1st Semester 1443H / Fall 2021**

**TCP-based client server application**

|  |  |
| --- | --- |
| **Student Name** | **ID** |
| Areej Turky Alotaibi | 440023303 |
| Raghad Adel Alshabana | 440021235 |
| Shoroog Saad Alarifi | 440022128 |
| Shoug Ali Alsuhaibani | 440022732 |

**Section:** 374

**Supervisor:** Basma Alsouli

**11th of December 2021**

**Table of Contents**

[Introduction 3](#_Toc90049844)

[Setting up the Programming Environment 4](#_Toc90049845)

[Steps for setting up the network 6](#_Toc90049846)

[Steps for Socket Programming 7](#_Toc90049847)

[Code and Comments 10](#_Toc90049848)

[Snapshots of the application outputs 14](#_Toc90049849)

[Problems we encountered, how did we fix them 16](#_Toc90049855)

[References 17](#_Toc90049856)

# Introduction

A client/server application model is often thought of as a high-powered computer unit located remotely that holds a big amount of data and has business logic to access it across a network.

The user interface is handled by the client software. This idea is not unique because any system that serves the request can be referred to as a server. Although the server waits for the client to initiate or start a conversation, in some cases the same program may act as both client and server. In that sense, a single machine can act as a network providing the communication between the client and the server program [1].

In this article, we will describe how to set up and develop a simple TCP-based client server application in java, also its communication over the network to exchange data.

# Setting up the Programming Environment

The main reason of choosing java language as our programming language is the familiarity that we used to with java, as all the past years we were working with it which increases our capability of creation and manipulation the code.

Moreover, that our devices have a Java Runtime Environment (JRE) installed already and Most types of computers will be compatible with a JRE too, where a Java program can run on it and make it easy for us to deal with.

Lastly, the language considered to be very simple it comes with a library of classes that offer commonly used utility functions that most Java programs can’t work without.

The IDE we worked with for this project is Visual Studio Code (VS), for MacOS downloading follow these Steps:

1. Visit this website and download the macOS version <https://code.visualstudio.com/download>
2. After the download is complete double click on the file and open it and you all set
3. In some cases, Visual Studio Code dose not end up In application all what you have to do is open your Download folder in finder and you will find it there after that just pull and drag the icon to the Application folder.

Snapshot of programming environment:

A screenshot of a computer

Description automatically generated

To compile and run the Program(on one host) we followed these steps:

1. Open the folder containing the codes in two separate terminal windows

Graphical user interface

Description automatically generated

**Step 1a**

Text

Description automatically generated

**Step 1b**

1. Compile and Run each of Server and Client code in the separate windows

Text

Description automatically generated

**Note** :In this project we did not install any libraries,because it was not needed.

# Steps for setting up the network

When we wanted to set up the hosts to communicate, we chose the wireless technology approach. So, to communicate the two hosts has to be on the same network after that we need the IP address of the host running the Server Code to put it as the IP address the Client Constructor receives.

To get the IP address of the host running the Server Code, we followed these Steps (For a Mac):

1. Open System Preferences
2. Go to Network
3. A screenshot of a computer

   Description automatically generated with medium confidenceWe found the IP of the device written under the Wi-Fi Status as shown

A screenshot of a computer

Description automatically generated with medium confidence After that we wrote the IP in the Client Code which was on a different host as shown below:

Then the connection was established successfully.

# Steps for Socket Programming

We use javasocket programming to implement the client- server communication over TCP protocol [1].

Steps for establishing a TCP socket on the **Client side**:

In general:

* Create a socket using the socket object.
* Connect the socket to the server by passing the Server’s IP address and a port number.
* Send and receive data by Initialize **DataInputStream** and **DataoutputStream** objects.
* Closing the connection by close() function that include in all objects we use.

In details:

* Create a socket using the **Socket** object.

private Socket socket = null;

* Initialize the socket with the Server’s IP address and a port number.

socket = new Socket(address, port);

* Receive data by Initialize **DataInputStream** object to take input form terminal using the Parameter System.in which indicates a standard input stream that is already open and ready to supply input data, typically this stream corresponds to keyboard input, or any input source specified by the host environment or user.

private DataInputStream input = null;

// takes input from terminal

input = new DataInputStream(System.in);

* Receive data by Initialize **DataInputStream** object to take input form server socket using the Parameter new BufferedInputStream(socket.getInputStream()) which Creates a BufferedInputStream and saves its argument (in this case a Returned input stream for the socket socket.getInputStream()), for later use. An internal buffer array is created and stored in.

private DataInputStream in = null;

// takes input from the server socket

in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

r

* Send data by Initialize **DataOutputStream** object to send output to the socket using the Parameter socket.getOutputStream() that Returns an output stream for the socket.

private DataOutputStream out = null;

// sends output to the socket

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

* Client will read the user’s typed in word from terminal using readLine()function in DataInputStream object.

line = input.readLine();

* Client will write data to server using writeUTF(data)function in DataOutputStream object.

out.writeUTF(line);

* Client will read data from server using readUTF()function in DataInputStream object.

line = in.readUTF();

* When the Client Closes the connection, all the input/output stream are closed and release any system resources associated with the stream.

input.close();

in.close();

out.close();

socket.close();

Steps for establishing a TCP socket on the **Server side**:

In general :

* Create a socket using the Serversocket object and pass the port number .
* Send and receive data by Initialize **DataInputStream** and **DataoutputStream** objects.
* accept the connection from client by accept() function using a socket object.
* Closing the connection by close() function that include in all objects we use.

In details:

* Create a socket using the **Serversocket** object.

private ServerSocket server = null;

* Initialize the **Serversocket** with a port number.

server = new ServerSocket(port);

* Receive data by Initialize **DataInputStream** object to take input from Client socket using the Parameter new BufferedInputStream(socket.getInputStream()) which Creates a BufferedInputStream and saves its argument (in this case a Returned input stream for the socket socket.getInputStream()), for later use. An internal buffer array is created and stored in.

private DataInputStream in = null;

// takes input from the client socket

in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

* Send data by Initialize **DataoutputStream** object to send output to Client socket using the Parameter socket.getOutputStream() that Returns an output stream for the socket.

private DataOutputStream out = null;

// sends output to the socket

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

* Accept a client connection using a **Socket** object and accept() function which Listens for a connection to be made to this socket and accepts it.

private Socket socket = null;

socket = server.accept();

* Server will read data from Client using readUTF()function in DataInputStream object.

line = in.readUTF();

* Server will write data to Client using writeUTF(data)function in DataOutputStream object.

out.writeUTF(line);

* When the Client Closes the connection, all the input/output stream are closed and release any system resources associated with the stream.

socket.close();

in.close();

out.close();

# Code and Comments

**Code of Server side:**

public class Server{

//initialize socket and input stream

private Socket socket = null;

private ServerSocket server = null;

private DataInputStream in = null;

private DataOutputStream out = null;

// constructor with port

public Server(int port){

// starts server and waits for a connection

try{

server = new ServerSocket(port);

System.out.println("Server started");

System.out.println("Waiting for a client ...");

socket = server.accept();

System.out.println("Client accepted");

System.out.println();

// takes input from the client socket

in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

// sends output to the socket

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

String line = "";

// reads message from client until "3" is sent

while (true){

try{

line = in.readUTF(); //reading from Client

if(line.equals("3"))

break;

System.out.println("Server Reflect: ");

if(line.startsWith("1")){

line = line.substring(1);

}else{ //Since there's no other choise, go option 2

line = line.substring(1);

String Encrypted = Alice(line,3); // encrypting the received word

System.out.println("Decrypted word:" + line);

System.out.print("Encrypted word:");

line = Encrypted;

}

System.out.println(line);

System.out.println();

out.writeUTF(line); //writing to Client

}

catch(IOException i){

System.out.println(i);

}

}

System.out.println("Closing connection");

// close connection

socket.close();

in.close();

out.close();

}

catch(IOException i){

System.out.println(i);

}

}

// Encryption Function

public static String Alice(String s , int key){

char[] chars=s.toCharArray();

for(int i=0 ; i< chars.length ; i++){

chars[i] +=key;

}

return String.valueOf(chars);

}

public static void main(String args[]){

Server server = new Server(500);

}

}

Q

**Code for Client side:**

public class Client{

// initialize socket and input output streams

private Socket socket = null;

private DataInputStream input = null;

private DataInputStream in = null;

private DataOutputStream out = null;

// constructor to put ip address and port

public Client(String address, int port){

// establish a connection

try{

socket = new Socket(address, port);

System.out.println("Connected");

// takes input from terminal

input = new DataInputStream(System.in);

// sends output to the socket

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

// takes input from the server socket

in = new DataInputStream(new BufferedInputStream(socket.getInputStream()));

}

catch(UnknownHostException u){

System.out.println(u);

}

catch(IOException i){

System.out.println(i);

}

// string to read message from input

String line = "";

char mode = '0';

// keep reading/writing until "3" is input

while (true){

System.out.println("1 Open mode");

System.out.println("2 Secure mode");

System.out.println("3 Quit");

try{

line = input.readLine();

mode = line.charAt(0);

out.writeUTF(line); //writing to Server

if(line.equals("3"))

break;

System.out.println("Client Received: ");

line = in.readUTF(); // reading from Server

if (mode == '2'){

String Decrypted = Bob(line,3); // decrypting the received word

System.out.println("Encrypted word:" + line);

System.out.print("Decrypted word:");

line = Decrypted;

}

System.out.println(line);

System.out.println();

}

catch(IOException i){

System.out.println(i);

}

}

System.out.println("Closing connection");

// close the connection

try{

input.close();

in.close();

out.close();

socket.close();

}

catch(IOException i){

System.out.println(i);

}

}

// Decryption Function

public static String Bob(String word , int key){

char[] chars=word.toCharArray();

for(int i=0 ; i< chars.length ; i++){

chars[i] -=key;

}

return String.valueOf(chars);

}

public static void main(String args[]){

Client client = new Client("127.0.0.1", 500);

}

}

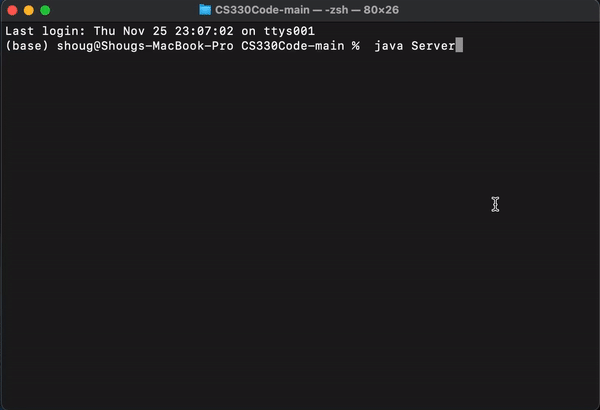
The structure of the code was acquired from here[2], with some modification to meet the project's requirements.

For a clearer version you can see here:

[https://github.com/CS330Code.git](https://github.com/alsuhaibanishoug/CS330Code.git)

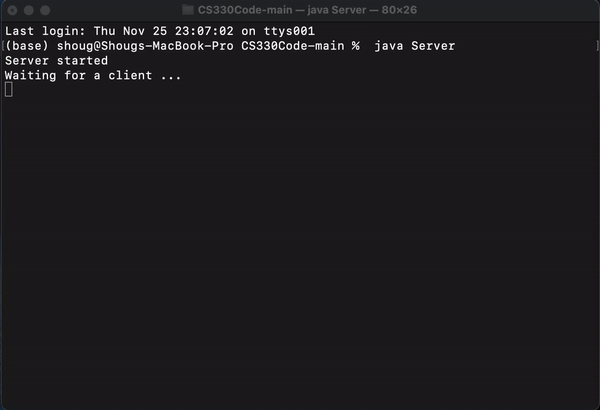
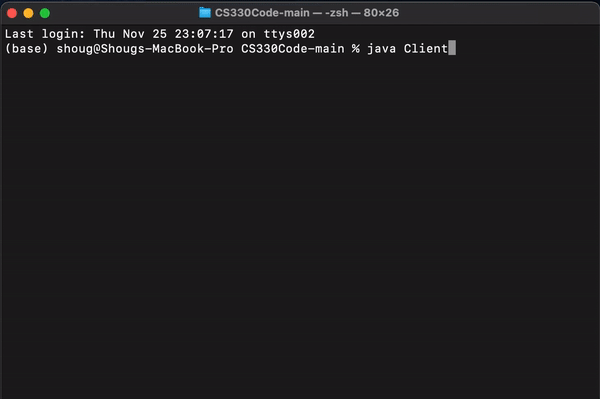
# Snapshots of the application outputs

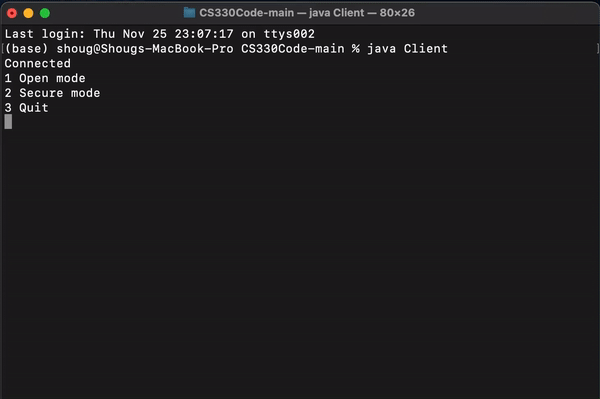
**Server Client**



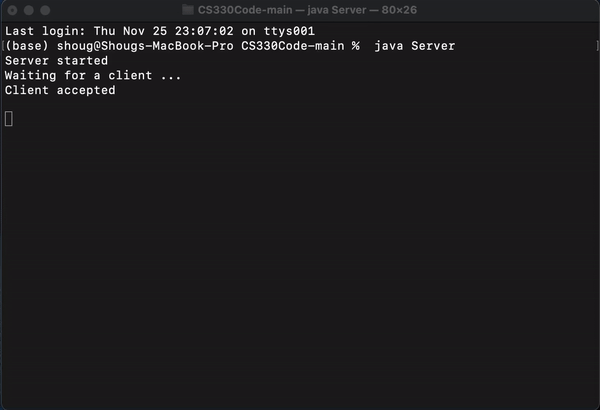
Not starting yet

Server is up no Client

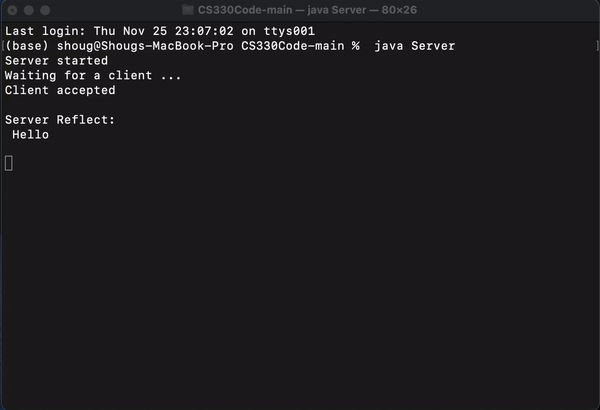
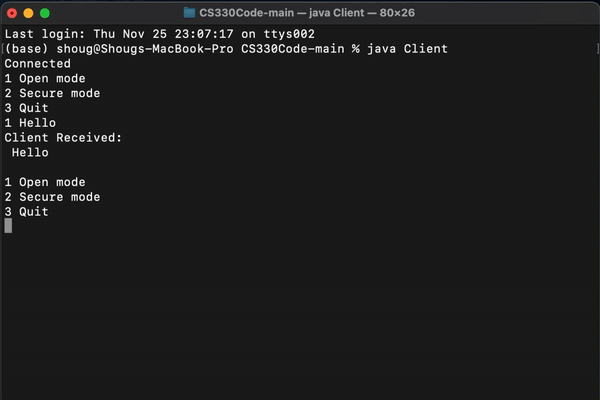




Server is up when Client requested a Connection



Client sends to Server in Open mode



Client sends to Server in Secure mode

# 

# Graphical user interface, text, application Description automatically generated

Client Quits Server’s session

Not starting yet

Server is down when Client requested a Connection

# Problems we encountered, how did we fix them

* **Problem1:** One of the problems we faced is how to exchange massages between two host devices not just two windows in terminal.

**Solution:** we solve it by take the IP of computer who represent the Server and adding in client file code.

* **Problem2:** Another problem we faced was in resending the massage the Server received back to the Client.

**Solution:** we solved it by defining a DataOutputStream object to send the message to the socket by it.

* **Problem3:** The problem was in the Client side also. We could not receive the message the Server send.

**Solution:** we solved it by defining another DataInputStream object different than the one we used to take the terminal input from, and used it to take the message from the socket.

# References

[1] <https://www.developer.com/web-services/socket-programming-tcp-client-server-application/>

[2] [https://www.geeksforgeeks.org/SocketProgramming](https://www.geeksforgeeks.org/socket-programming-in-java/)