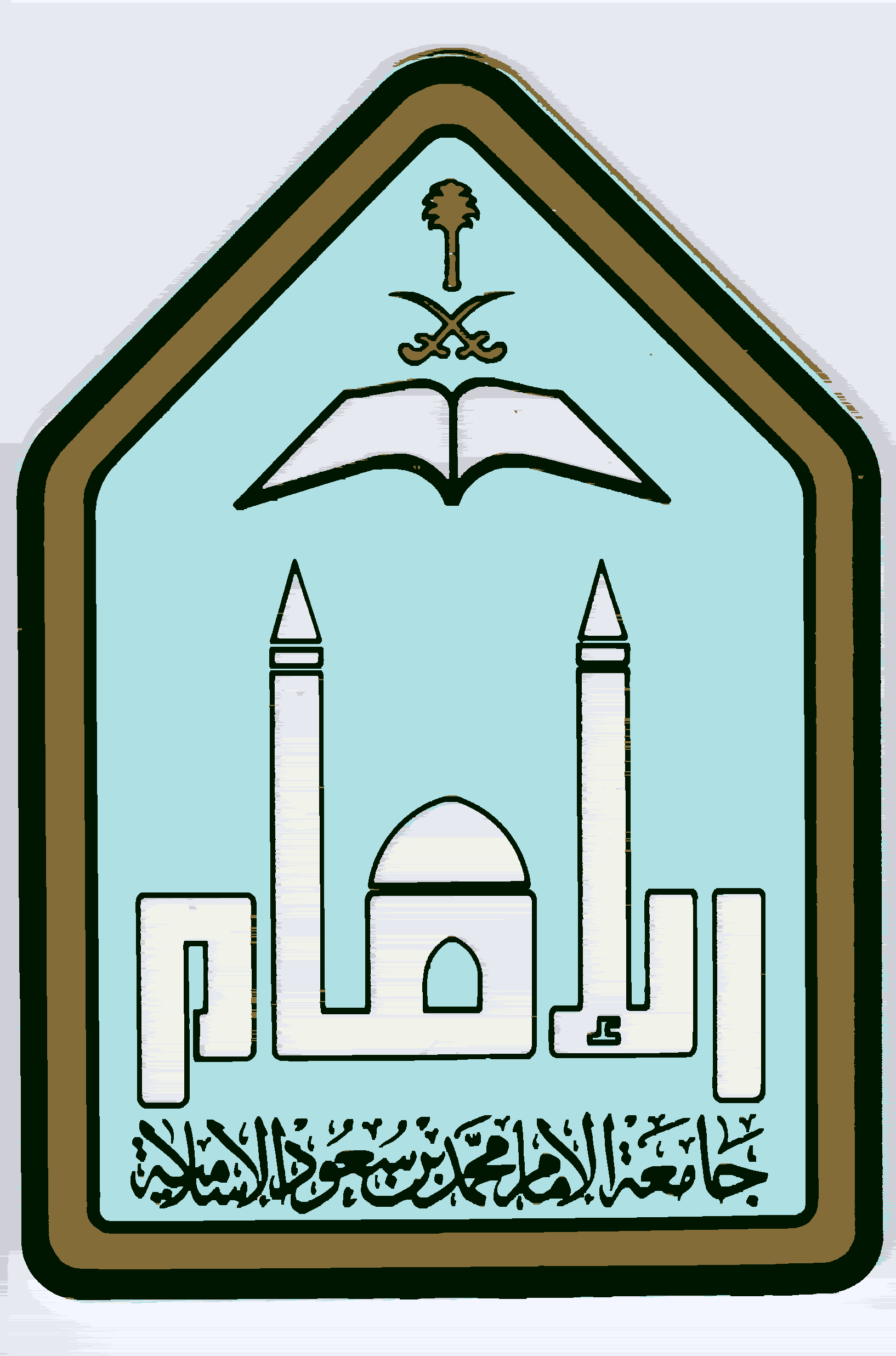
****



**CS330–Computer Networks Project**

**Client/Server Application**

**Option 1**

**Students Names:**

**Yasser Al-Ruwaiti-440017590**

**Abdulrahman Al-Rubayan-440017248**

**Abdulkarim Alluhaydan-440016617**

**Faisal Al-zaydan-440013811**

**Section: 171-173**

Date of Submission: 5/20/2022

**Instructor's Name:**

Dr. Abdulaziz I. Almohimeed

**1-Setting up the Programming Environment**

1-To start off, we chose Java as our programming language. The main reason was due to the lack of time that we’re facing and since we all are very familiar with it, you can add up to that the huge number of resources regarding socket programming using java.

2-First you need to download AndroidStudio: <https://developer.android.com/studio>

After installing it you need to create a new device from the Device manager: A screenshot of a computer

Description automatically generated

And use Pixel XL to use it in the emulator.

After that all you have to do is to run the server’s class separately and then client 1, and then client 2.

3-We have used several libraries, such as:

**java.util,** contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array):

**java.util.Scanner,** a simple text scanner which can parse primitive types and strings using regular expressions.

**java.util.HashMap,** hash table-based implementation of the Map interface**.** [1]

**java.net**, Provides the classes for implementing networking applications:

**java.net.Inet6Address**, this class represents an Internet Protocol version 6 (IPv6) address.

**java.net.InetSocketAddress,** this class implements an IP Socket Address (IP address + port number) It can also be a pair (hostname + port number), in which case an attempt will be made to resolve the hostname.

**java.net.Socket,** this class implements client sockets (also called just "sockets").

**java.net.ServerSocket,** this class implements server sockets.

**java.net.UnknownHostException,** [2]

**java.io**, Provides for system input and output through data streams, serialization and the file system:

**java.io.BufferedReader,** reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

**java.io.BufferedWriter,** writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings. [3]

**androidx.appcompat.app.AppCompatActivity;**

**android.os.Bundle;**

**android.view.View;**

**android.widget.TextView;**

**2-Steps for TCP/UDP socket programming for client-server connection:**

**We have used java socket programming to implement the client-server communication over TCP protocol.**

**We have used the TCP/IP for couple of reasons:**

In our case, the program serves a very sensitive kind of users, kids. Thus, it’s very important to our program to be let’s say fault tolerant, since the TCP/IP provides a reliable connection setup between the client-server architecture, and also it uses feedbacking which will come in very handy in case of any packet loss. You don’t want the child to freak out or gets frustrated.

Our message format for the TCP/IP is consisting of 3 digits for the ID followed by a comma ‘,’ followed by “out” therefore it’s 7 bytes in length.

**We used this code of multiple clients chatting to make it work with our problem**[4]

1-The server will create the socket using:

ServerSocket serverSocket = new ServerSocket(1234);

Server server = new Server(serverSocket);

server.startServer();

We are creating a TCP server socket with port number = 1234 we are going to create an instance of the type server and pass the serverSocket to the constructor of the Server’s class then we are going to call the function startServer in the Server class to start the server.

startServer()

the function startServer it will loop until the serverSocket closes it, it will create an object of type ClientHandler for every new connection from the clients and start a separate thread for every

client.

Class ClientHandler

public ClientHandler(Socket socket)

this.socket = socket; this.bufferedReader=newBufferedReader(newInputStreamReader(socket.getInputStream()));

this.bufferedWriter=new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

this.clientName = bufferedReader.readLine();

this.clientID = bufferedReader.readLine();

clientHandlers.put(this.clientID,this)

The constructor of ClientHandler will accepts a socket, after that we will create a buffer reader and writer for that socket and we will wait for the client to identify himself by his ID, after that they will be added to the list of clientHandlers.

public void run()

The function will be running on a separate thread because it’s listening for massages from other clients, and while it’s listening it will be blocking the program until a message is received.

2-The client:

public class Client {

private Socket socket;

private BufferedReader bufferedReader;

private BufferedWriter bufferedWriter;

private String name;

private String ID;

The client will be able to communicate with the server via the Socket attribute socket, and sends and receives messages using the two buffers, bufferedreader and bufferedwriter, and each client is identified by his name and ID in the server.

public void setClient()

This function will basically send an identification message to the server to inform it with the client’s name and ID.

public void sendAlert(String message)

This function will send an alert which is a flag message to change the state in the UI.

The message is the ID of the client we want to reach.

public void listenForAlert()

What this function does is to create a separate thread which will be waiting for an alert from other clients, once it receives an alert it changes the state in the UI.

**3-Steps for setting up the network**

In order to connect our two clients, client 1 being the parent’s phone and client 2 is the kid’s smart-watch, we must use some sort of a handler to process their requests. So, we have implemented a Server class in java which will be handling the requests, works as follows:

The server will be on a specific machine, for example Abdulrahman’s PC.

Yasser will represent the parent’s phone, and Faisal will represent the kid’s watch.

Each located at their house.

**1-**Abdulrahman will specify a specific port number and then will run the server’s class.

**2-**client 1 being Yasser will have to match the port number and the IPv6 of Abdulrahman’s PC and then will run the client’s class.

**3-**Faisal will do the same as Yasser; now, the server’s class uses the ClientHandler class which will be responsible for managing the clients within the server.

**We clearly have used unguided media, for the sake of practicability.**

I mean imagine a plugged wire running through the whole school :)

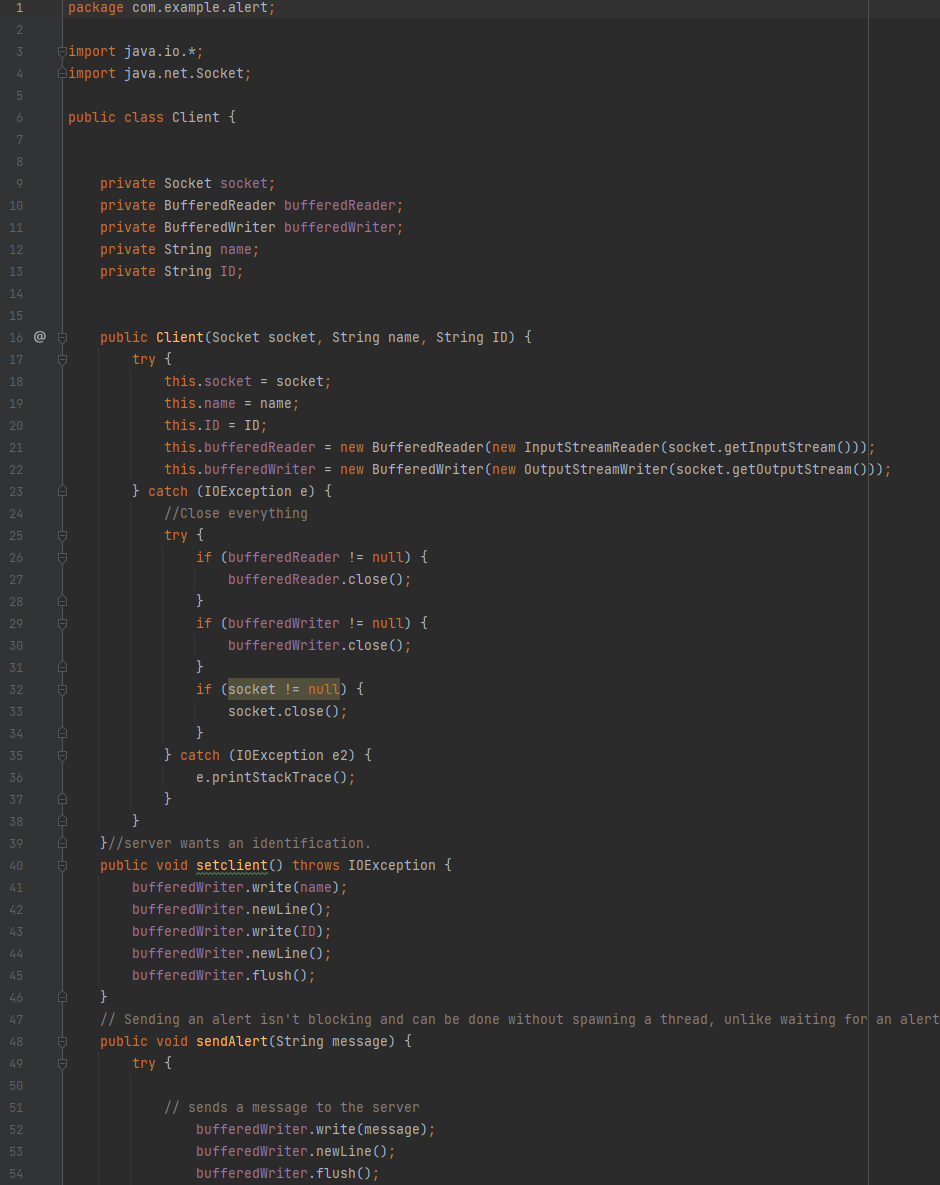
**4- Codes and comments:**

**Code of server side:**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Code for client side:**

**Text

Description automatically generated**

**5- Snapshots of the application outputs.**

We’ve relied on Yasser’s PC to run the server’s code.

Abdulrahman smart phone to be the kid’s phone.

And lastly the parent’s phone was implemented on an emulator.

Port# 1234.

Now once you run the app(Client 1 Parent):

A screenshot of a phone

Description automatically generated with low confidence

On the emulator(Client 2 Kid):

Shape, square

Description automatically generated

After sending an alert from the parent’s phone:

A picture containing square

Description automatically generated

After sending an alert from the kid’s device:

A picture containing square

Description automatically generated

1. **Problems and solutions:**

**Problem 1:** We struggled to make our code work on different networks i.e., to connect the client 1 being for instance Yasser’s device to the server located at Abdulkarim’s house.

**Solution:**  The code was only working on the local-host, we had to improve it in which it can

support the IPv6, after that the communication setup was possible.

**Problem 2:** There was an issue regarding the implementation of the code in AndroidStudio in the client’s side, it wouldn’t work for when we run the program for some reason.

**Solution:** We’ve discovered after a long time that the problem was that we didn’t extends AsyncTask abstract class. [5]

**Problem 3:** A problem arises each time we click on the alert button, it creates a new client every time.

**Solution:** A different type of handling the alert sending was built, we divided the process into two methods, setConnection and sendAlert. Before that we were only using one method that does the work, now works perfectly fine.

**Unsolved Problems.**

**Problem:** When we wanted to add an alert sound, we’ve encountered a problem. The problem was that we wanted to use a method which will apply the sound when an alert appears, however, the method is in the client’s class and we wanted to call it in the Activity class, but the program won’t work properly.

**Problem:** We wanted to implement the application on the Wear OS, however, we faced some issues, it wasn’t working as intended, crashes a lot + we didn’t have a watch to even test it on.

**Deployment Process:**

In order to release the application to the public we must enhance the software up a bit:

Design a login system in which users can create accounts and add their kids in it.

Implement a database to handle the data from the users such as their usernames, passwords

and ID’s.

Use a hosting service instead of using one of our PCs to play the role of a server.

Design an application logo.

After that we’re probably all set to just drop it in the market.

Since we used java, it only works on android OS therefore we can drop it on Play Store.

**References:**

[1] <https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>

[2] <https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html>

[3] <https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html>

[4] <https://wittcode.com/java/java-socket-programming-multiple-clients-chat>

[5] <https://www.youtube.com/watch?v=29y4X65ZUwE>