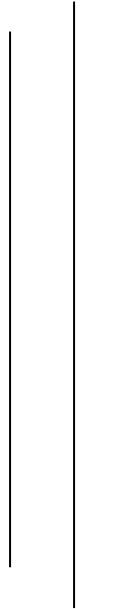




SUNWAY

INT'L BUSINESS SCHOOL



Program Name: BCS

Course Code: CSC 2513

Course Name: Programming Fundamentals

Project Work

Date of Submission: 13th May2021

Submitted by:

Student Name: Udit Kumar Mahato

IUKL ID: 042002900006

Semester: Second Semester

Intake: Sept 2020

Submitted To:

Faculty Name: Mr. Prakash Chandra

Department: PO Office (BCS)

A CHAT APPLICATION USING SOCKETS IN JAVA.

CONTENTS

	Page no
1. Chapter 1: Abstract	4
2. Chapter 2: Introduction	5 - 10
3. Chapter 3: Testing	11 -15
4. Chapter 4: Output	16
5. Appendix	17-33
6. References	34

CHAPTER 1

ABSTRACT

This report presents a details overview in developing Multi Client Single Server based chat application using socket programming. The application is developed using Java programing to share messages and files between clients and server. The primary objective of this report is to present the principles behind socket programming and the libraries available for socket programming applications in Java.

The application uses a specific port to communication between each other to share the messages and files. In this application, Multiple clients can be created through this application and can share message both privately and publicly.

CHAPTER TWO

INTRODUCTION

This application is designed to be Single server multi-client chat application including File transfer both publicly and privately and ` consists of a server.java and a client.java files representing the client and server programs of the chat application. Server program uses TCP connection protocol to listen for clients connecting to its socket using the port specified by the user. If the user doesn't specify any port, then the server runs on port 1234.

Socket Overview

The socket is an object that represents a low-level access point to the IP stack. This socket can be opened or closed or one of a set number of intermediate states. A socket can send and receive data down disconnection. Data is generally sent in blocks of few kilobytes at a time for efficiency; each of these blocks are called a packet.

All packets that travel on the internet must use the Internet Protocol. This means that the source IP address, destination address must be included in the packet. Most packets also contain a port number. A port is simply a number between 1 and 65,535 that is used to differentiate higher protocols. Ports are important when it comes to programming your own network applications because no two applications can use the same port.

This project can be mainly divided into two modules:

1. Server

2. Client

This project is mainly depended on client/server model. The client requests the server and server responses by granting the client's request. The proposed system should provide both of the above features along with the followed ones:

1.Server

A server is a computer program that provides services to other computer programs (and their users) in the same or other computers. The computer that a server program runs in is also frequently referred to as a server. That machine may be a dedicated server or used for other purposes as well. Example Server, Database, Dedicated, Fileserver, Proxy Server, Web Server. The server is always waiting for client requests. The client comes and goes but the server remains the same.

A server application normally listens to a specific port waiting for connection requests from a client. When a connection request arrives, the client and the server establish a dedicated connection over which they can communicate. During the connection process, the client is assigned a local port number, and binds a socket to it. The client talks to the server by writing to the socket and gets information from the server by reading from it. Similarly, the server gets a new local port number (it needs a new port number

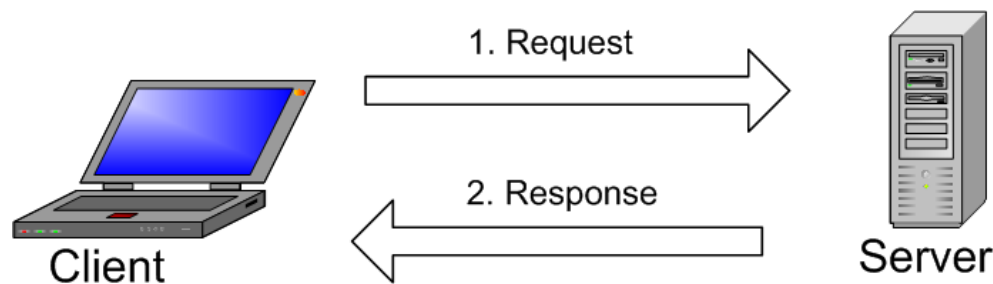
so that it can continue to listen for connection requests on the original port). The server also binds a socket to its local port and communicates with the client by reading from and writing to it. The client and the server must agree on a protocol that is, they must agree on the language of the information transferred back and forth through the socket. Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request.

The steps in establishing a socket on server side are as follows: –

- Create a socket with the `socket ()` system call.
- Bind the socket to an address using the `bind ()` system call. For a server socket on the Internet, an address consists of a port number on the host machine.
- Listen for connections with the `listen ()` system call.
- Accept a connection with the `accept ()` system call. This call typically blocks the connection until a client connects with the server.
- Send and receive data using the `read ()` and `write ()` system calls.

2. Client

On the client site the client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

- Create a socket with the `socket ()` system call.
- Connect the socket to the address of the server using the `connect ()` system call.

- Send and receive data. There are a number of ways to do this, but the simplest way is to use the `read ()` and `write ()` system calls.
- simple Client Program in Java
- The steps for creating a simple client program are:
 - Create a Socket Object: `Socket client = new Socket (server, portside);`
 - Create I/O streams for communicating with the
 - `server. input = new DataInputStream (client. getInputStream ());`
 - `output = new DataOutputStream (client. getOutputStream ());`
 - Perform I/O or communication with the server:
 - Receive data from the server: `String line = input. readLine ();`
 - Send data to the server: `output. writeBytes ("Hello Sunway\n");`
 - Close the socket when done:
- `client.close();`

Execution Steps for Server.java

- Open Command Line and navigate to the folder where the file `Server.java` is present.
- Type the command `javac Server.java` for compilation of the the Server.
- Type the command `java Server 8000` to execute the program where 8000 is the port.
- The server displays a message "Server is running using specified port number=8000". This message proves that the server program is running and is waiting for clients to connect to this server.
- If the user doesn't specify any port, then the Server uses default port of 1234.

Execution Steps for Client.java

- Open Command Line and navigate to the folder where the file `Client.java` is present.
- Type the command `javac Client.java` for compilation of the the Server.
- Type the command `java Client localhost 8000` to execute the program where localhost is the server's name and 8000 is the port.
- The client program prompts for the username.

- Once the client enters the name and hits enter, the server displays a message "Client 1 is connected!" confirming that the Client1 is connected to the server.

Also, the client is prompted as

*** Welcome Client1 to our chat room ***

Enter /quit to leave the chat room

New Receiving file directory for this client created!!

Please Enter command:

Similarly, other clients can be connected to the server using the same Client.java program. Please note that if the testing of the program is done in a single machine, then there should be separate folders created for server and each client.

If the user doesn't specify any port, then the Client uses localhost and default port of 1234.

Functionalities performed by this application

1. Broadcast:

This feature enables a user to send a message or a file to all other clients connected to the server. For sending file a keyword sendfile is used followed by the filename.

Example for sending message:

Please Enter command:

Hello

Example for sending file:

Please Enter command:

sendfile test.pdf

2. Unicast:

This feature enables a user to send a message or a file to a particular client connected to the server.

For sending message the following format should be followed: -

@<TargetUsername>:<your message>

Example for sending message:

Please Enter command:

@Client1:Hello

For sending file the following format should be followed: -

@<TargetUsername>:sendfile <your filename with path or without path>

Example for sending file:

Please Enter command:

@Client1:sendfile test.pdf

3. Block Cast:

This feature enables a user to send a message or a file to an all clients except a particular client connected to the server.

For sending message the following format should be followed: -

!<TargetUsername>:<your message>

Example for sending message:

Please Enter command:

! Client1: Hello

For sending file the following format should be followed: -

!<TargetUsername>:sendfile <your filename with path or without path>

Example for sending file:

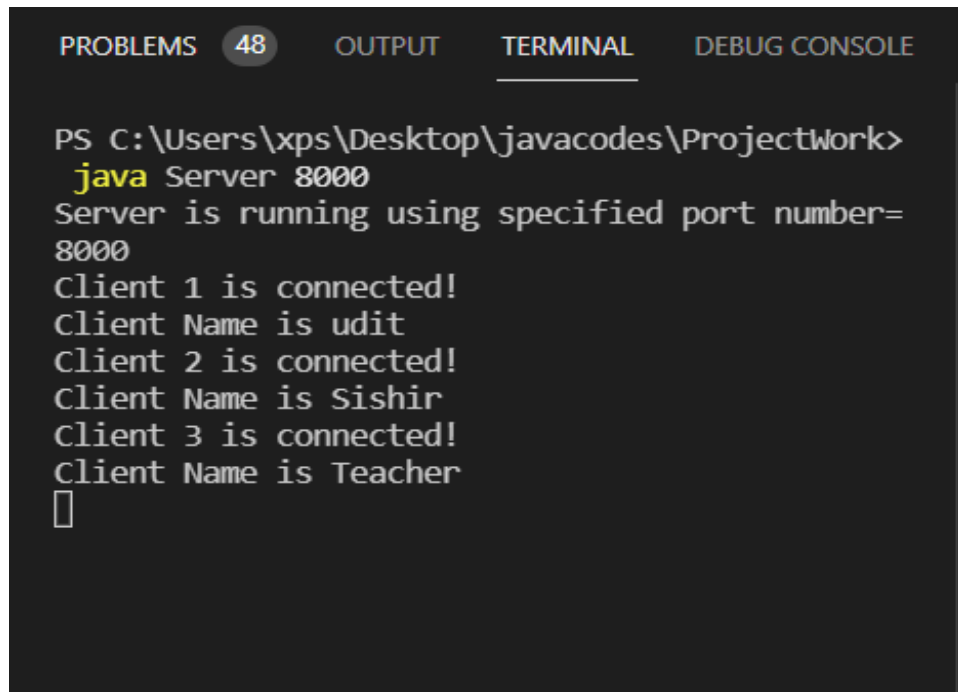
Please Enter command:

!Client1: sendfile test.pdf

CHAPTER THREE

TESTING

I created a server specifying a port 8000



```
PROBLEMS 48 OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
  java Server 8000
Server is running using specified port number=
8000
Client 1 is connected!
Client Name is udit
Client 2 is connected!
Client Name is Sishir
Client 3 is connected!
Client Name is Teacher
█
```

Then created three CLIENT named : udit, sishir and Teacher

<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : udit *** Welcome udit to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: Sishir has joined Teacher has joined </pre>	<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : Sishir *** Welcome Sishir to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: Teacher has joined </pre>	<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : Teacher *** Welcome Teacher to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: </pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Then broadcasted MESSAGE among both the students from teacher.

<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : udit *** Welcome udit to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: Sishir has joined Teacher has joined <Teacher> Hello Students </pre>	<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : Sishir *** Welcome Sishir to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: Teacher has joined <Teacher> Hello Students </pre>	<pre> PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000 Server: localhost, Port: 8000 Please enter your name : Teacher *** Welcome Teacher to our chat room *** Enter /quit to leave the chat room Receiving file directory for this client already exists!! Please Enter command: Hello Students Broadcast message sent successfully. Please Enter command: </pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Then tried sending private message only to udit from teacher using function unicast

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
Teacher
*** Welcome Teacher to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Hello Students
Broadcast message sent successfully.
Please Enter command:
@udit: Have you done your project?
Private Message sent to udit
Please Enter command:

```

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
udit
*** Welcome udit to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Sishir has joined
Teacher has joined
<Teacher> Hello Students
<Teacher> Have you done your project?

```

Then sent file named hello.pdf to everyone from teacher

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
Teacher
*** Welcome Teacher to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Hello Students
Broadcast message sent successfully.
Please Enter command:
@udit: Have you done your project?
Private Message sent to udit
Please Enter command:
sendfile hello.pdf
Broadcast file sent successfully
Please Enter command:

```

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
  java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
udit
*** Welcome udit to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Sishir has joined
Teacher has joined
<Teacher> Hello Students
<Teacher> Have you done your project?
File Received.

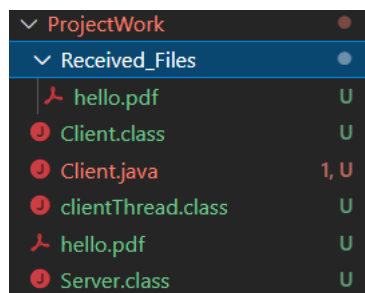
```

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork>
  java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
Sishir
*** Welcome Sishir to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Teacher has joined
<Teacher> Hello Students
File Received.

```

Now the sent file should already been saved in Received files



Now tried broadcasting message from teacher to everyone connected except udit using Block Cast

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
Teacher
*** Welcome Teacher to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Hello Students
Broadcast message sent successfully.
Please Enter command:
@udit: Have you done your project?
Private Message sent to udit
Please Enter command:
sendfile hello.pdf
Broadcast file sent successfully
Please Enter command:
!udit:hello
>>Blockcast message sent to everyone except udit
Please Enter command:

```

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Server 8000
Server is running using specified port number=8000
Client 1 is connected!
Client Name is udit
Client 2 is connected!
Client Name is Sishir
Client 3 is connected!
Client Name is Teacher
Broadcast message sent by Teacher
Teacher transferred a private message to client udit
Broadcast file sent by Teacher
Message sent by Teacher to everyone except udit

```

```

host 8000
Server: localhost, Port: 8000
Please enter your name :
udit
*** Welcome udit to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Sishir has joined
Teacher has joined
<Teacher> Hello Students
<Teacher> Have you done your project?
File Received.

```

```

PS C:\Users\xps\Desktop\javacodes\ProjectWork> java Client localhost 8000
Server: localhost, Port: 8000
Please enter your name :
Sishir
*** Welcome Sishir to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Teacher has joined
<Teacher> Hello Students
File Received.
<Teacher> hello

```

CHAPTER FOUR

OUTPUT

Screenshots of output

Server:

```
ied port number=8000
Client 1 is connected!
Client Name is udit
Client 2 is connected!
Client Name is Sishir
Client 3 is connected!
Client Name is Teacher
Broadcast message sent by Teacher
Teacher transferred a private message to client udit
Broadcast file sent by Teacher
Message sent by Teacher to everyone except udit
█
```

Client: Udit

```
udit to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Sishir has joined
Teacher has joined
<Teacher> Hello Students
<Teacher> Have you done your project?
File Received.
```

```
hi
Broadcast message sent successfully.
Please Enter command:
█
```

Client: Teacher

```
PS C:\Users\xps\Desktop\javac
java Client
Default Server: localhost, De
0
Please enter your name :
Teacher
*** Welcome Teacher to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Hello Students
Broadcast message sent successfully.
Please Enter command:
@udit: Have you done your project?
Private Message sent to udit
Please Enter command:
sendfile hello.pdf
Broadcast file sent successfully
Please Enter command:
!udit:hello
>>Blockcast message sent to everyone except udit
Please Enter command:
<udit> hi
█
```

Client: Sishir

```
r your name :
Sishir
*** Welcome Sishir to our chat room ***
Enter /quit to leave the chat room
Receiving file directory for this client already exists!!
Please Enter command:
Teacher has joined
<Teacher> Hello Students
File Received.
<Teacher> hello
<udit> hi
█
```


APPENDIX

SOURCE CODE

SERVER

```
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.ArrayList;

import java.net.ServerSocket;

/*
 * A chat server that delivers public and private messages and files.
 */
public class Server {

    // The server socket.
    private static ServerSocket serverSocket = null;
    // The client socket.
    private static Socket clientSocket = null;

    public static ArrayList<clientThread> clients = new ArrayList<clientThread>();
;

    public static void main(String args[]) {

        // The default port number.
        int portNumber = 1234;

        if (args.length < 1)
        {

            System.out.println("No port specified by user.\nServer is running using default port number=" + portNumber);

        }
        else
        {

```

```

        portNumber = Integer.valueOf(args[0]).intValue();

        System.out.println("Server is running using specified port number=" +
portNumber);
    }

    /*
     * Open a server socket on the portNumber (default 1234).
     */
    try {
        serverSocket = new ServerSocket(portNumber);
    } catch (IOException e) {
        System.out.println("Server Socket cannot be created");
    }

    /*
     * Create a client socket for each connection and pass it to a new client
     * thread.
     */

    int clientNum = 1;
    while (true) {
        try {

            clientSocket = serverSocket.accept();
            clientThread curr_client = new clientThread(clientSocket, client
s);

            clients.add(curr_client);
            curr_client.start();
            System.out.println("Client " + clientNum + " is connected!");
            clientNum++;

        } catch (IOException e) {

            System.out.println("Client could not be connected");
        }

    }

}

}
/*

```

```

    * This client thread class handles individual clients in their respective thread
    S
    * by opening a separate input and output streams.
    */
class clientThread extends Thread {

    private String clientName = null;
    private ObjectInputStream is = null;
    private ObjectOutputStream os = null;
    private Socket clientSocket = null;
    private final ArrayList<clientThread> clients;
    public clientThread(Socket clientSocket, ArrayList<clientThread> clients) {

        this.clientSocket = clientSocket;
        this.clients = clients;

    }

    public void run() {

        ArrayList<clientThread> clients = this.clients;

        try {
            /*
             * Create input and output streams for this client.
             */
            is = new ObjectInputStream(clientSocket.getInputStream());
            os = new ObjectOutputStream(clientSocket.getOutputStream());

            String name;
            while (true) {

                synchronized(this)
                {
                    this.os.writeObject("Please enter your name :");
                    this.os.flush();
                    name = ((String) this.is.readObject()).trim();

                    if ((name.indexOf('@') == -1) || (name.indexOf('!') == -1)) {
                        break;
                    } else {
                        this.os.writeObject("Username should not contain '@' or '
! ' characters.");
                        this.os.flush();

```

```

    }
}

/* Welcome the new the client. */

System.out.println("Client Name is " + name);

this.os.writeObject("*** Welcome " + name + " to our chat room **
*\nEnter /quit to leave the chat room");
this.os.flush();

this.os.writeObject("Directory Created");
this.os.flush();
synchronized(this)
{
    for (clientThread curr_client : clients)
    {
        if (curr_client != null && curr_client == this) {
            clientName = "@" + name;
            break;
        }
    }

    for (clientThread curr_client : clients) {
        if (curr_client != null && curr_client != this) {
            curr_client.os.writeObject(name + " has joined");
            curr_client.os.flush();

        }
    }
}

/* Start the conversation. */

while (true) {

    this.os.writeObject("Please Enter command:");
    this.os.flush();

    String line = (String) is.readObject();

```

```

        if (line.startsWith("/quit")) {

            break;
        }

        /* If the message is private sent it to the given client. */

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

            unicast(line,name);

        }

        /* If the message is blocked from a given client. */

        else if(line.startsWith("!"))
        {
            blockcast(line,name);
        }

        else
        {

            broadcast(line,name);

        }

    }

    /* Terminate the Session for a particluar user */

    this.os.writeObject("*** Bye " + name + " ***");
    this.os.flush();
    System.out.println(name + " disconnected.");
    clients.remove(this);

    synchronized(this) {

        if (!clients.isEmpty()) {

            for (clientThread curr_client : clients) {

```

```

        if (curr_client != null && curr_client != this && curr_client.clientName != null) {
            curr_client.os.writeObject("*** The user " + name + " disconnected ***");
            curr_client.os.flush();
        }
    }
}

this.is.close();
this.os.close();
clientSocket.close();

} catch (IOException e) {

    System.out.println("User Session terminated");

} catch (ClassNotFoundException e) {

    System.out.println("Class Not Found");

}

}

/**** This function transfers message or files to all the client except a particular client connected to the server ****/

void blockcast(String line, String name) throws IOException, ClassNotFoundException {

    String[] words = line.split(":", 2);

    /* Transferring a File to all the clients except a particular client */

    if (words[1].split(" ")[0].toLowerCase().equals("sendfile"))
    {
        byte[] file_data = (byte[]) is.readObject();
    }
}

```

```

        synchronized(this) {
            for (clientThread curr_client : clients) {
                if (curr_client != null && curr_client != this && curr_client
.clientName != null
                        && !curr_client.clientName.equals("@"+words[0].substr
ing(1)))
                    {
                        curr_client.os.writeObject("Sending_File:"+words[1].split
(" ",2)[1].substring(words[1].split("\\s",2)[1].lastIndexOf(File.separator)+1));
                        curr_client.os.writeObject(file_data);
                        curr_client.os.flush();
                    }
            }

            /* Echo this message to let the user know the blocked file was se
nt.*/

            this.os.writeObject(">>Blockcast File sent to everyone except "+w
ords[0].substring(1));
            this.os.flush();
            System.out.println("File sent by "+ this.clientName.substring(1)
+ " to everyone except " + words[0].substring(1));
        }
    }

    /* Transferring a message to all the clients except a particular client */
/

    else
    {
        if (words.length > 1 && words[1] != null) {
            words[1] = words[1].trim();
            if (!words[1].isEmpty()) {
                synchronized (this){
                    for (clientThread curr_client : clients) {
                        if (curr_client != null && curr_client != this && cur
r_client.clientName != null
                                && !curr_client.clientName.equals("@"+words[0
].substring(1))) {
                            curr_client.os.writeObject("<" + name + "> " + wo
rds[1]);
                            curr_client.os.flush();
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    /* Echo this message to let the user know the blocked mes
sage was sent.*/

        this.os.writeObject(">>Blockcast message sent to everyone
except "+words[0].substring(1));
        this.os.flush();
        System.out.println("Message sent by " + this.clientName.su
bstring(1) + " to everyone except " + words[0].substring(1));
    }
}
}
}

/**** This function transfers message or files to all the client connected to
the server */

void broadcast(String line, String name) throws IOException, ClassNotFoundExc
ption {

    /* Transferring a File to all the clients */

    if (line.split("\\s")[0].toLowerCase().equals("sendfile"))
    {

        byte[] file_data = (byte[]) is.readObject();
        synchronized(this){
            for (clientThread curr_client : clients) {
                if (curr_client != null && curr_client.clientName != null &&
curr_client.clientName!=this.clientName)
                {
                    curr_client.os.writeObject("Sending_File:"+line.split("\\
s",2)[1].substring(line.split("\\s",2)[1].lastIndexOf(File.separator)+1));
                    curr_client.os.writeObject(file_data);
                    curr_client.os.flush();
                }
            }

            this.os.writeObject("Broadcast file sent successfully");
            this.os.flush();
            System.out.println("Broadcast file sent by " + this.clientName.su
bstring(1));

```



```

    }
}

else
{
    /* Transferring a message to all the clients */

    synchronized(this){

        for (clientThread curr_client : clients) {

            if (curr_client != null && curr_client.clientName != null &&
curr_client.clientName!=this.clientName)
            {

                curr_client.os.writeObject("<" + name + "> " + line);
                curr_client.os.flush();

            }

            this.os.writeObject("Broadcast message sent successfully.");
            this.os.flush();
            System.out.println("Broadcast message sent by " + this.clientName
.substring(1));
        }

    }

}

/***** This function transfers message or files to a particular client connect
ed to the server ****/

void unicast(String line, String name) throws IOException, ClassNotFoundException {

    String[] words = line.split(":", 2);

    /* Transferring File to a particular client */

    if (words[1].split(" ")[0].toLowerCase().equals("sendfile"))
    {
        byte[] file_data = (byte[]) is.readObject();
    }
}

```

```

        for (clientThread curr_client : clients) {
            if (curr_client != null && curr_client != this && curr_client.clientName != null
                && curr_client.clientName.equals(words[0]))
            {
                curr_client.os.writeObject("Sending_File:"+words[1].split(" ", 2)[1].substring(words[1].split("\\s", 2)[1].lastIndexOf(File.separator)+1));
                curr_client.os.writeObject(file_data);
                curr_client.os.flush();
                System.out.println(this.clientName.substring(1) + " transferred a private file to client " + curr_client.clientName.substring(1));

                /* Echo this message to let the sender know the private message was sent.*/

                this.os.writeObject("Private File sent to " + curr_client.clientName.substring(1));
                this.os.flush();
                break;
            }
        }

        /* Transferring message to a particular client */

        else
        {
            if (words.length > 1 && words[1] != null) {

                words[1] = words[1].trim();

                if (!words[1].isEmpty()) {

                    for (clientThread curr_client : clients) {
                        if (curr_client != null && curr_client != this && curr_client.clientName != null
                            && curr_client.clientName.equals(words[0])) {
                            curr_client.os.writeObject("<" + name + "> " + words[1]);
                            curr_client.os.flush();

```



```

private static boolean closed = false;

public static void main(String[] args) {

    // The default port.
    int portNumber = 1234;
    // The default host.
    String host = "localhost";

    if (args.length < 2) {
        System.out.println("Default Server: " + host + ", Default Port: " + portNumber);
    } else {
        host = args[0];
        portNumber = Integer.valueOf(args[1]).intValue();
        System.out.println("Server: " + host + ", Port: " + portNumber);
    }

    /*
     * Open a socket on a given host and port. Open input and output streams.
     */
    try {
        clientSocket = new Socket(host, portNumber);
        inputLine = new BufferedReader(new InputStreamReader(System.in));
        os = new ObjectOutputStream(clientSocket.getOutputStream());
        is = new ObjectInputStream(clientSocket.getInputStream());
    } catch (UnknownHostException e) {
        System.err.println("Unknown " + host);
    } catch (IOException e) {
        System.err.println("No Server found. Please ensure that the Server program is running and try again.");
    }

    /*
     * If everything has been initialized then we want to write some data to the
     * socket we have opened a connection to on the port portNumber.
     */
    if (clientSocket != null && os != null && is != null) {
        try {

            /* Create a thread to read from the server. */
            new Thread(new Client()).start();
            while (!closed) {

```

```

        /* Read input from Client */

        String msg = (String) inputLine.readLine().trim();

        /* Check the input for private messages or files */

        if ((msg.split(":").length > 1))
        {
            if (msg.split(":")[1].toLowerCase().startsWith("sendfile"
))
            {
                File sfile = new File((msg.split(":")[1]).split(" ",2
)[1]);

                if (!sfile.exists())
                {
                    System.out.println("File Doesn't exist!!");
                    continue;
                }

                byte [] mybytearray = new byte [(int)sfile.length()]
;

                FileInputStream fis = new FileInputStream(sfile);
                bis = new BufferedInputStream(fis);
                while (bis.read(mybytearray,0,mybytearray.length)>=0)
                {
                    bis.read(mybytearray,0,mybytearray.length);
                }
                os.writeObject(msg);
                os.writeObject(mybytearray);
                os.flush();

            }
            else
            {
                os.writeObject(msg);
                os.flush();
            }

        }

        /* Check the input for broadcast files */

        else if (msg.toLowerCase().startsWith("sendfile"))
        {

```

```

        File sfile = new File(msg.split(" ",2)[1]);

        if (!sfile.exists())
        {
            System.out.println("File Doesn't exist!!");
            continue;
        }

        byte [] mybytearray = new byte [(int)sfile.length()];
        FileInputStream fis = new FileInputStream(sfile);
        bis = new BufferedInputStream(fis);
        while (bis.read(mybytearray,0,mybytearray.length)>=0)
        {
            bis.read(mybytearray,0,mybytearray.length);
        }
        os.writeObject(msg);
        os.writeObject(mybytearray);
        os.flush();

    }

    /* Check the input for broadcast messages */

    else
    {
        os.writeObject(msg);
        os.flush();
    }

}

/*
 * Close all the open streams and socket.
 */
os.close();
is.close();
clientSocket.close();
} catch (IOException e)
{
    System.err.println("IOException: " + e);
}

```

```

    }
}

/*
 * Create a thread to read from the server.
 */

public void run() {
    /*
     * Keep on reading from the socket till we receive "Bye" from the
     * server. Once we received that then we want to break.
     */
    String responseLine;
    String filename = null;
    byte[] ipfile = null;
    FileOutputStream fos = null;
    BufferedOutputStream bos = null;
    File directory_name = null;
    String full_path;
    String dir_name = "Received_Files";

    try {

        while ((responseLine = (String) is.readObject()) != null) {

            /* Condition for Directory Creation */

            if (responseLine.equals("Directory Created"))
            {
                //Creating Receiving Folder

                directory_name = new File((String) dir_name);

                if (!directory_name.exists())
                {
                    directory_name.mkdir();

                    System.out.println("New Receiving file directory for this
client created!!");
                }

                else
                {

```

```

        System.out.println("Receiving file directory for this cli
ent already exists!!");
    }
}

/* Condition for Checking for incoming files */

else if (responseLine.startsWith("Sending_File"))
{
    try
    {
        filename = responseLine.split(":")[1];
        full_path = directory_name.getAbsolutePath()+"/"+filename
;

        ipfile = (byte[]) is.readObject();
        fos = new FileOutputStream(full_path);
        bos = new BufferedOutputStream(fos);
        bos.write(ipfile);
        bos.flush();
        System.out.println("File Received.");
    }
    finally
    {
        if (fos != null) fos.close();
        if (bos != null) bos.close();
    }
}

/* Condition for Checking for incoming messages */

else
{
    System.out.println(responseLine);
}

/* Condition for quitting application */

if (responseLine.indexOf("*** Bye") != -1)

    break;
}

```



```
        closed = true;
        System.exit(0);

    } catch (IOException | ClassNotFoundException e) {

        System.err.println("Server Process Stopped Unexpectedly!!");

    }

}
```

REFERENCES

<http://java.sun.com/docs/books/tutorial/networking/sockets/>

<http://makemobiapps.blogspot.com/p/multiple-client-server-chat-programming.html>

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

<https://www.codejava.net/java-se/networking/how-to-create-a-chat-console-application-in-java-using-socket>

<https://mail.codejava.net/java-se/networking/java-socket-client-examples-tcp-ip>

<https://www.codejava.net/java-se/networking/upload-files-by-sending-multipart-request-programmatically>

<https://www.codejava.net/java-se/networking/java-socket-client-examples-tcp-ip>

<https://www.youtube.com/watch?v=WeaB8pAGIDw&t=13s>

<https://www.youtube.com/watch?v=6oBePDVZHYc>