COMP2221: NETWORKS
COURSEWORK 2

DESCRIPTION

The definition of a communication protocol is a system that is compiled of rules in telecommunications which allow for two or more entities of a communications system to transfer information via kind of variation of physical quantity. Communicating system use well defined formats (protocols) for exchanging various messages, files and folders.
Sockets allow communication between two different processes on the same machine or the different machines.
It follows standard Unix file descriptor, where all I/O actions are undertaken by the process of reading/writing the descriptor. The descriptor can be described as an integer related to an open file.

Sockets are probably the most commonly used type of communication transport protocol over TCP/IP that you will use. TCP sockets provide you with a reliable, nearly error-free data pipe between two endpoints, both of which can send and receive streams of bytes back and forth, without data being lost or duplicated.

- Server sockets are bound to well-known endpoints where an endpoint is the (protocol, address, port) tuple. The endpoint is formed by the protocol specified in the socket() system call and the addressing information specified in the bind() system call.
- When the server calls the listen() system call, the network stack creates a queue in which pending connections are placed into. A hint to the size of the queue is given as the backlog parameter to listen().
- The server then calls accept() to pull a new connection from the queue.
- Client sockets send messages to server sockets by specifying the server endpoint in a call to connect() and then sending data using send() or write().
- When the client calls connect(), a connection is pushed onto the server-side queue where it sits until the server accepts the connection.

This type of connection between devices is also sometimes referred to as a client/server model. One device, known as the client, creates a socket, connects to the server, and then begins sending and receiving data. On the other side, the server creates a socket and listens for an incoming connection from the client. Once a connection is initiated, the server accepts the connection, and then starts to send and receive data to and from the incoming client. The data that the client and server send back and forth is completely up to you; however, several well-known communication protocols have already been established, such as HTTP or FTP.

The application is made up of 3 files as listed. I have also given a brief description of the roles of each file and their purpose in the application as a whole.

- **FileServer.java**
  Libraries : java.io, java.net.* , java.util.concurrent.Executors
  java.util.concurrent.ExecutorService

  Class: Creating the class for the server which starts the server by using server socket of given port.

  Method: Main method which allow to start server using server socket with port '4444' .Calls the another class 'Connection with client' to accept the connection with client socket and perform the all specified task given in the coursework.

- **ConnectionWithClient.java**
  Libraries: java.io.BufferedOutputStream, java.io.FileOutputStream, java.io.IOException, java.io.InputStream, java.net.Socket, java.util.Scanner java.io.*, java.net.*, java.util.logging.Level, java.util.logging.Logger, java.util.zip.ZipEntry, java.util.zip.ZipOutputStream, java.util.zip.ZipInputStream (using * in case any important library missed while coding)

  Class: Create the class to connect the client and server (ConnectionWithClient) which implements Runnable class/interface.

  Method: -receiveFile() method to receive the file from client.
  -sendFile() method to sent the file to client.
  -zip() method to zip the folder and sent the whole folder to client by calling the zip() in sendFile() method.
  -unzip() method to unzip the folder sent by the server. This method is called in receiveFile().

- **FileClient.java**
  Libraries: java.io.BufferedOutputStream, java.io.FileOutputStream, java.io.IOException, java.io.InputStream, java.net.Socket, java.util.Scanner java.io.*, java.net.*, java.util.logging.Level, java.util.logging.Logger,

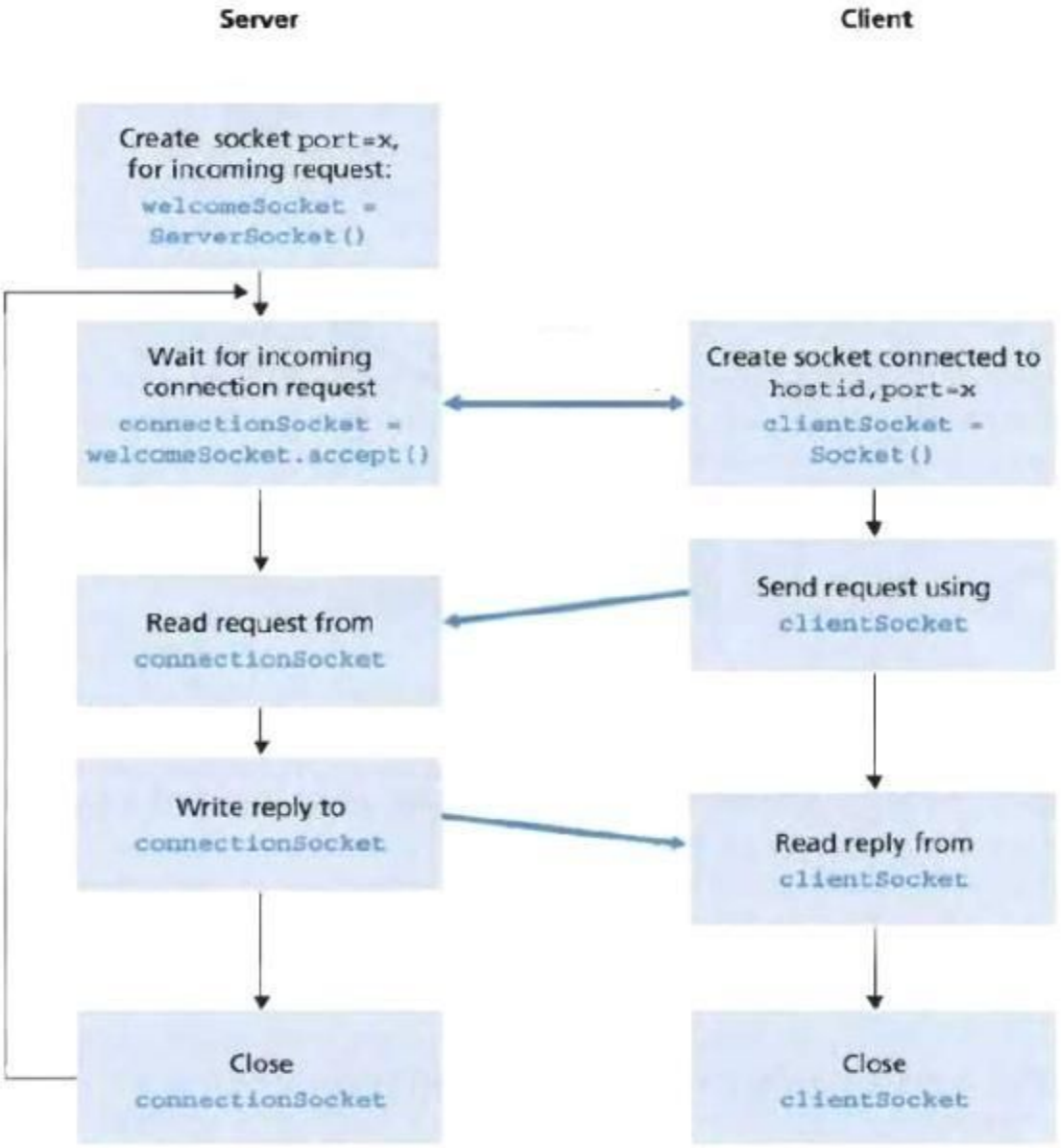
java.util.zip.ZipEntry, java.util.zip.ZipOutputStream, java.util.zip.ZipInputStream (using * in case any important library missed while coding)

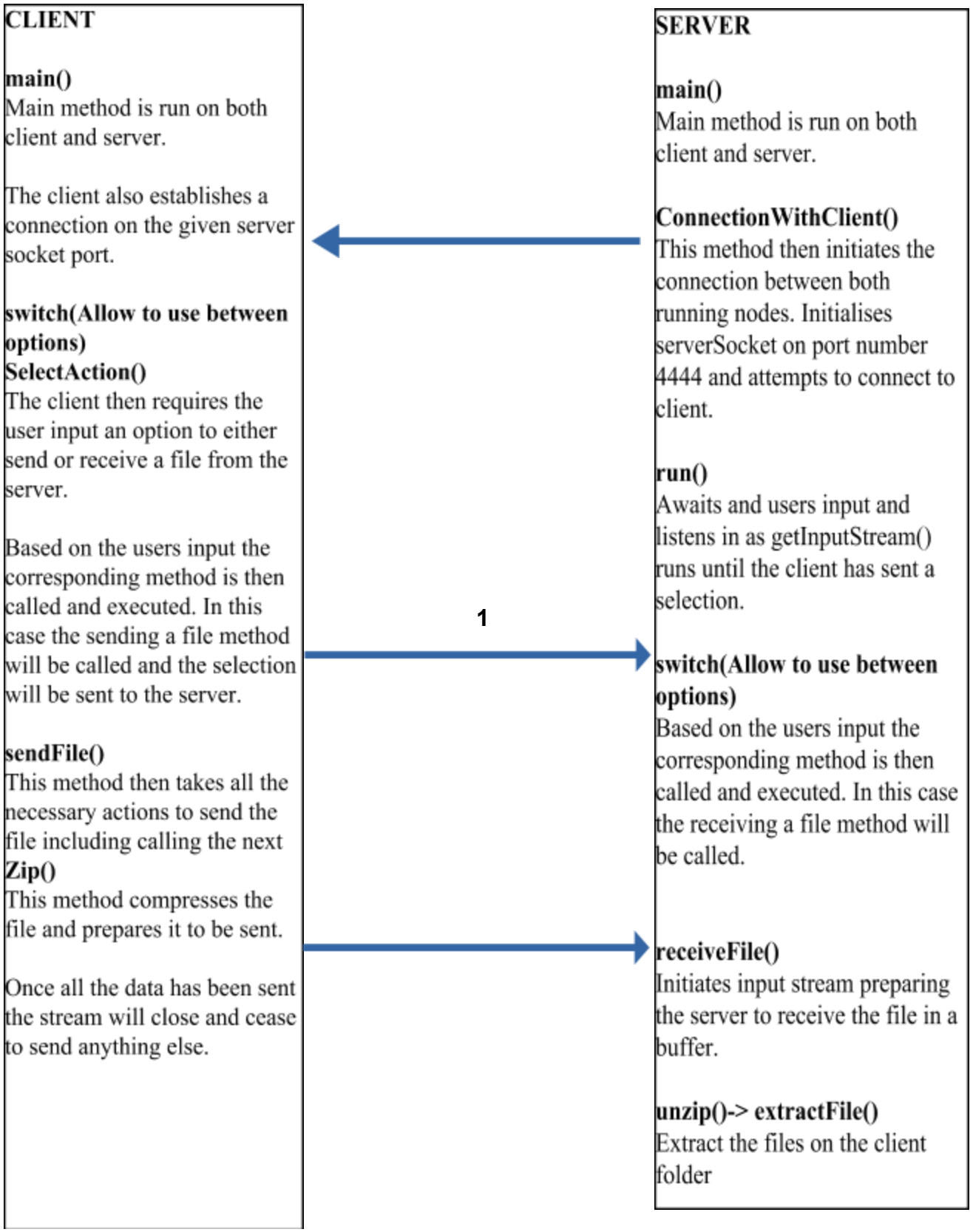
Class: Creating the class for the client which allow the server connection.

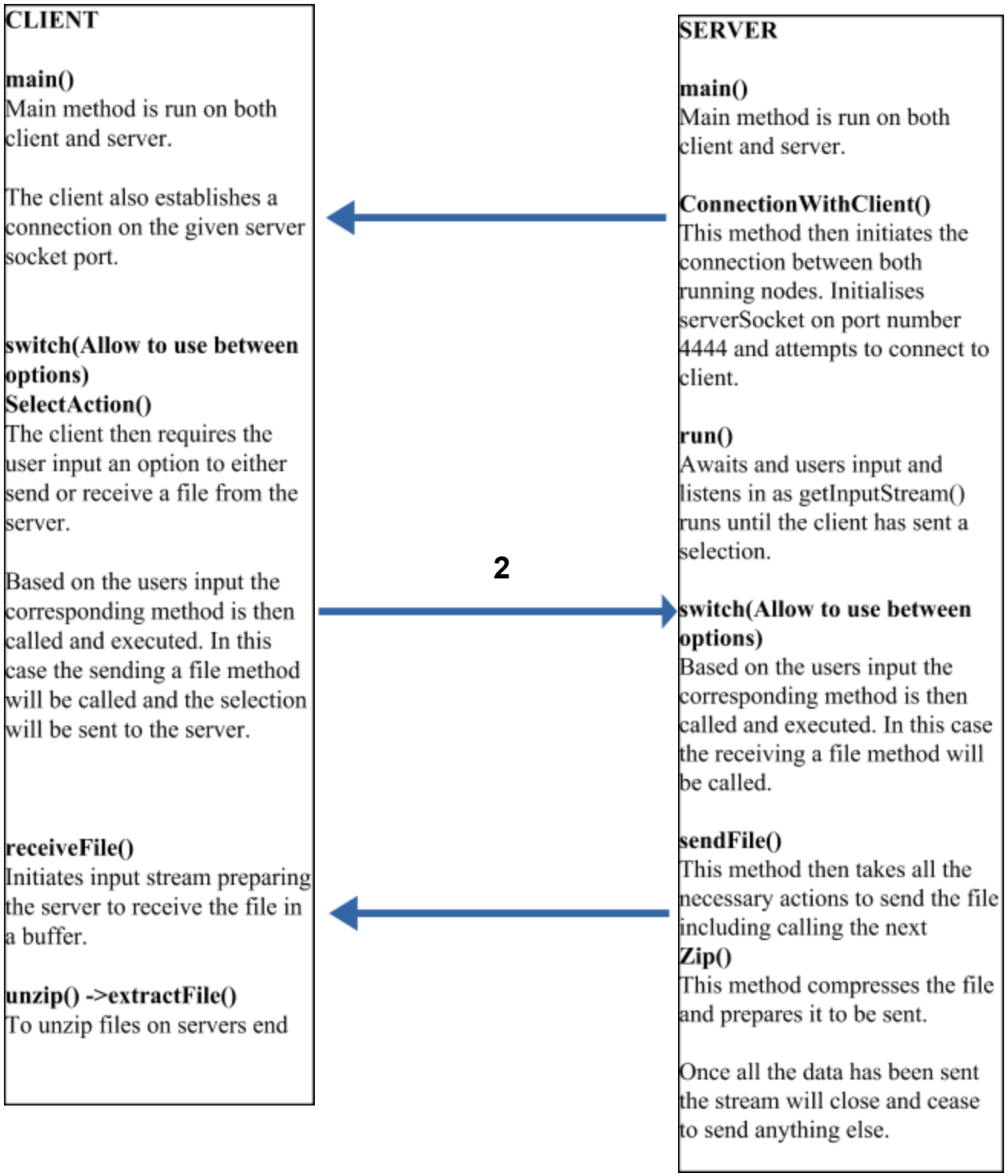
Method: -main() which allow to server connection and which also uses selectAction() method to display the correct result of the given option like view files, send files or receive files .
-selectAction() method which allow user to select the option.
-sendFile() method which sent the file or folder to the server
-receiveFile() method to receive the file from server.
-zip() method to zip the folder and sent the whole folder to server by calling the zip() in sendFile() method.
-unzip() method to unzip the folder sent by the server. This method is used in receiveFile().

DIAGRAMS WITH EXPLANATION

First the Server creates a socket for each socket the client has.
Then the server waits for a client call in order to obtain and therefore accept the network communication and once that connection is established then a request is sent (send or receive files).
Lastly the request is received and can be accepted (receive) or rejected and block to further cancel any incoming connections.

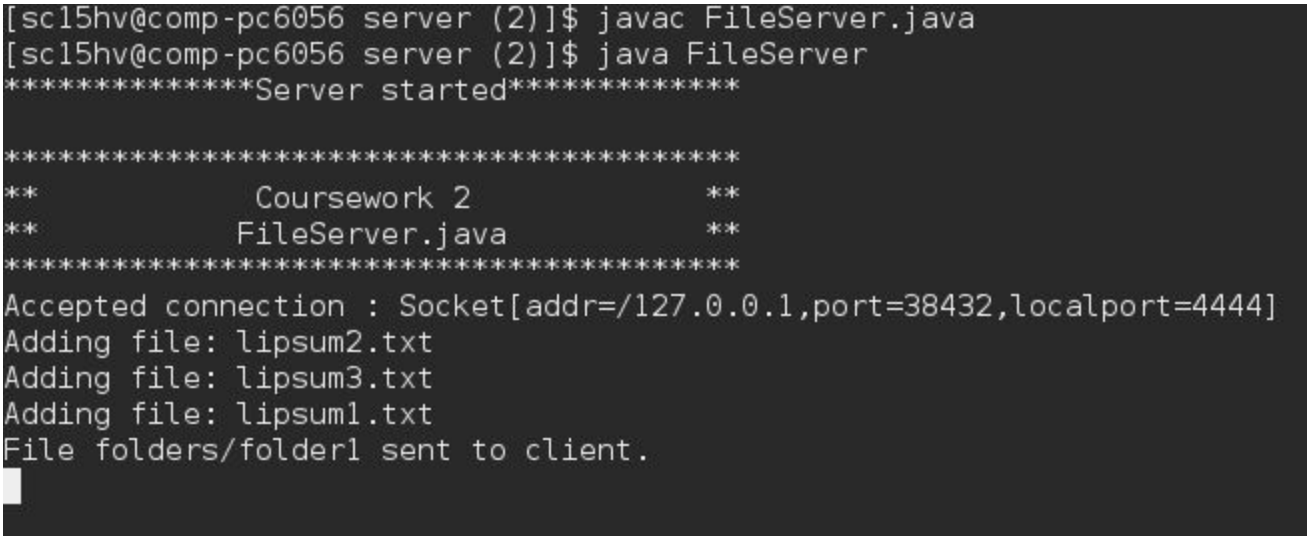



CLIENT
main()
Main method is run on both client and server.
The client also establishes a connection on the given server socket port.
switch(Allow to use between options)
SelectAction()
The client then requires the user input an option to either send or receive a file from the server.
Based on the users input the corresponding method is then called and executed. In this case the sending a file method will be called and the selection will be sent to the server.
sendFile()
This method then takes all the necessary actions to send the file including calling the next
Zip()
This method compresses the file and prepares it to be sent.
Once all the data has been sent the stream will close and cease to send anything else.
SERVER
main()
Main method is run on both client and server.
ConnectionWithClient()
This method then initiates the connection between both running nodes. Initialises serverSocket on port number 4444 and attempts to connect to client.
run()
Awaits and users input and listens in as getInputStream() runs until the client has sent a selection.
1
switch(Allow to use between options)
Based on the users input the corresponding method is then called and executed. In this case the receiving a file method will be called.
receiveFile()
Initiates input stream preparing the server to receive the file in a buffer.
unzip()-> extractFile()
Extract the files on the client folder

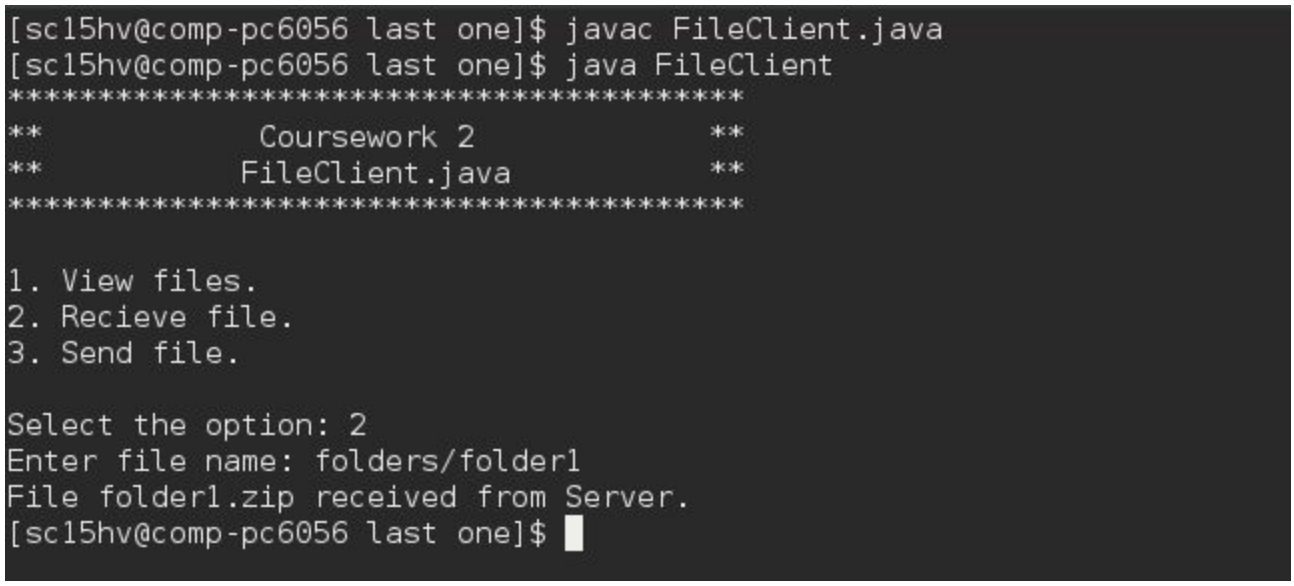

CLIENT
main()
Main method is run on both client and server.
The client also establishes a connection on the given server socket port.
switch(Allow to use between options)
SelectAction()
The client then requires the user input an option to either send or receive a file from the server.
Based on the users input the corresponding method is then called and executed. In this case the sending a file method will be called and the selection will be sent to the server.
receiveFile()
Initiates input stream preparing the server to receive the file in a buffer.
unzip() ->extractFile()
To unzip files on servers end
SERVER
main()
Main method is run on both client and server.
ConnectionWithClient()
This method then initiates the connection between both running nodes. Initialises serverSocket on port number 4444 and attempts to connect to client.
run()
Awaits and users input and listens in as getInputStream() runs until the client has sent a selection.
2
switch(Allow to use between options)
Based on the users input the corresponding method is then called and executed. In this case the receiving a file method will be called.
sendFile()
This method then takes all the necessary actions to send the file including calling the next
Zip()
This method compresses the file and prepares it to be sent.
Once all the data has been sent the stream will close and cease to send anything else.

WORKING

Server



Client



REFERENCES

Writing the server side of sockets:
https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html

Socket Examples:
https://www.cs.uic.edu/~troy/spring05/cs450/sockets/socket.html

Network Sockets:
https://en.wikipedia.org/wiki/Network_socket

More java socket examples:
http://iprodeveloper.com/rpg-programming/java-sockets-server-example

More Socket Tutorials:
http://o7planning.org/en/10393/java-socket-programming-tutorial

Multi Threaded socket based server:
http://www.kieser.net/linux/java_server.html

Simple server client program:
http://www.nakov.com/inetjava/lectures/part-1-sockets/InetJava-1.4-TCP-Sockets.html

More information on Java sockets:
https://www.codeproject.com/Tips/991180/Java-Sockets-and-Serialization

Guide to understanding Network Programming using sockets:
http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html

TCP/IP Illustrated: The implementation, Volume 2
W. Richard Stevens, Gary R. Wright