ReadMe File

Server (Multi-threaded) and Client

**Name: Sagar Sharma**

**UTA ID: 1001626958**

***Environment Used***

1. Java SE Development Kit 8 Update 181
2. Google Chrome stable versions - 60.0.3112.78
3. Windows command prompt
4. The default port used for communication is 8080

***Java and txt files used***

1. All files should remain in the same directory
2. Server.java (Server code)
3. Client.java (Client code)
4. Earth.txt (the file used for transmission)
5. Index.html(the file set to be default filename when no filename is passed)
6. File names are case sensitive

*Running* ***Server.java*** *on windows command prompt*

*Since java looks for* ***class path environment variable*** *set up, it might give a “****could not load main class****” error when ran from cmd, so to avoid this error please use these commands below in its entirety.*

*First go in the* ***same directory*** *as all the files for the project are in and follow these commands:*

1. **javac Server.java** (this command will compile the Server.java file and create a new Server.class file)
2. **java -cp ./classes;. Server 8080** (this command looks for the Server.class file in the same directory as the .java file and passes the 8080 as the default port)

**OR**

1. **java -cp ./classes;. Server** (this command looks for the Server.class file in the same directory as the .java file and has port number to be fixed to 8080 already)

*Running* ***Client.java*** *on windows command prompt*

1. **javac Client.java** (this command will compile the Client.java file and create a new Client.class file)
2. **java -cp ./classes;. Client localhost 8080 earth.txt** (this command looks for the Client.class file in the same directory as the .java file and passes the **127.0.0.1** as IP address,**8080** as the default port and **earth.txt** for the file to be transmitted)

**OR**

1. **java -cp ./classes;. Client localhost 8080 index.html** (this command looks for the Client.class file in the same directory as the .java file and passes the **127.0.0.1** as IP address,**8080** as the default port and **index.html** for the file to be transmitted)

**OR**

1. **java -cp ./classes;. Client localhost** (this command looks for the Client.class file in the same directory as the .java file and sets **127.0.0.1** as IP address for localhost,**8080** as the default port and sets **index.html** as filename for the file to be transmitted)
2. **wrong file name** gives a 404 NOT FOUND ERROR
3. **wrong port number** raises an *error*. *Connection Refused*

Running Client in Browser

1. [**http://localhost:8080/earth.txt**](http://localhost:8080/earth.txt)
2. **http://localhost:8080/index.html**
3. **wrong file name** gives a 404 NOT FOUND ERROR
4. **wrong port number** raises an *error*.

**References**

1. Skeleton code provided for Java Server Code
2. https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
3. https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html
4. https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
5. https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html
6. https://docs.oracle.com/javase/7/docs/api/java/nio/file/Path.html
7. https://docs.oracle.com/javase/7/docs/api/java/nio/file/Paths.html
8. https://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html
9. https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html
10. https://docs.oracle.com/javase/7/docs/api/java/io/FileNotFoundException.html
11. https://docs.oracle.com/javase/8/docs/api/java/io/InputStream.html
12. https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html
13. https://docs.oracle.com/javase/7/docs/api/java/io/OutputStream.html
14. https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html
15. https://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html
16. https://docs.oracle.com/javase/7/docs/api/java/io/DataInputStream.html
17. https://docs.oracle.com/javase/7/docs/api/java/io/DataOutputStream.html
18. ***https://en.wikipedia.org/wiki/Earth (this text has been used in the earth.txt file)***

Server Side when correct command is passed in cmd:

![A screenshot of a cell phone

Description generated with high confidence]()

Client Side when passed correct command in cmd with correct parameters

***Connection parameters and the file data with RTT (Client-Side Logs)***

earth.txt file being passed as filename

![A screenshot of a cell phone

Description generated with very high confidence]()

Index.html being passed as filename

![A screenshot of a cell phone

Description generated with high confidence]()

No filename or port number being passed; 8080 and index.html taken as the default port number and filename

![A screenshot of a cell phone

Description generated with very high confidence]()

Server Side when passed correct parameters at the Client Side

***Server-Side logs (Server-Side connection Parameters)***

![A screenshot of a cell phone

Description generated with high confidence]()

Command from browser and result from a correct command

earth.txt passed as the filename

![A screenshot of a social media post

Description generated with very high confidence]()

Index.html passed as the filename

![A screenshot of a social media post

Description generated with very high confidence]()

Server Side with an improper file name command from Client Side

***Server-Side Logs (specified file cannot be found)***

![A screenshot of a cell phone

Description generated with high confidence]()

Client Side with improper file name command in command prompt

***Client-Side Logs (404 NOT FOUND ERROR)***

![A screenshot of a cell phone

Description generated with very high confidence]()

Client Side with improper file name command in browser

***404 NOT FOUND ERROR***

![A screenshot of a social media post

Description generated with very high confidence]()

***CODE DOCUMENTATION***

***Server-Side Code***

**//import all the built-in classes to be used in the program from java**

import java.net.ServerSocket; **// implements server sockets**

import java.net.Socket;  **//implements client sockets**

import java.nio.file.Path; **//represent a system dependent file path**

import java.nio.file.Paths; **// return a Path by converting a path string or URI**

import java.util.StringTokenizer; **//for breaking strings into Tokens**

import java.io.FileInputStream;**//obtains input bytes from a file in a file system**

import java.io.FileNotFoundException;**// Signals that an attempt to open the file denoted by a specified pathname has failed**

import java.io.InputStream; **//representing an input stream of bytes**

import java.io.InputStreamReader; **// reads bytes and decodes them into characters using a specified charset**

import java.io.OutputStream; **//represents an output stream of bytes**

import java.net.InetAddress; **// represents an Internet Protocol (IP) address**

import 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**

import java.io.DataInputStream; **// lets an application read primitive Java data types from an underlying input stream in a machine-independent way**

import java.io.DataOutputStream; **// lets an application write primitive Java data types to an output stream in a portable way**

**// defining Server class and initializing variables**

public class Server

{

**//main**

public static void main(String args[]) throws Exception{

int port\_numb = 0; **//initializing to zero(port number for the communication)**

if(args.length<1)

{

port\_numb = 8080; **//if length of command in cmd is less than 1;set the port number to be default 8080**

}

else

{

port\_numb = Integer.parseInt(args[0]); **//otherwise port number is set to number passed in the cmd**

**}**

ServerSocket Socket\_server = new ServerSocket(port\_numb); **//connecting socket to the network and server is ready to receive client's http request messages**

System.out.println("Server is waiting for client's http requests....");

System.out.println("----------------------------------------------------------------------------------");

**//client's http request messages handling**

while(true)//indefinite loop

{

Socket server = Socket\_server.accept(); **//connection requests for TCP connection by client handling**

HttpRequest request = new HttpRequest(server); **//object to handle client's http request**

Thread thread = new Thread(request); **//thread to handle client's http requests**

thread.start(); **// thread initiates**

}

}

public static class HttpRequest implements Runnable{ **// implementing multithreading for server**

final static String CRLF = "\r\n"; **//CRLF = carriage return line feed**

Socket socket;

public HttpRequest(Socket socket) throws Exception{ **////initializing constructor for client's http request**

this.socket = socket;

}

private static String format\_type(String file\_name) **//stating the format of the file to be served to the client**

{ if(file\_name.endsWith(".htm") || file\_name.endsWith(".html") || file\_name.endsWith(".txt")) **//if text or html file format**

return "text/html";

else if(file\_name.endsWith(".jpeg") || file\_name.endsWith(".jpg")) **//if picture formats//if jpeg or jpg**

**return "image/jpeg";**

else if(file\_name.endsWith(".gif") )

return "image/gif"; **//if gif format**

return "some uncommon file format"; **//other file format**

}

public void run() **//For interface:runnable//run() method**

{

try

{

request\_process();

}

catch (Exception error)

{

System.out.println(error);

}

}

private static void bytes\_send(FileInputStream f\_i\_s, OutputStream os) throws Exception{

byte[] buffer = new byte[1024]; **// creating 1024 bytes sized buffer**

int bytes = 0;

while((bytes = f\_i\_s.read(buffer))!=-1) **// client file request sent to socket's output stream**

{

os.write(buffer, 0, bytes);

}

}

private void request\_process() throws Exception

{

InputStream is = new DataInputStream(socket.getInputStream()); **//I/O stream reference for Socket**

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

BufferedReader b\_r = new BufferedReader(new InputStreamReader(is)); **//input stream**

String line\_header = ""; **//initializing null values for line\_header and requested data lines**

String line\_request = "";

int line\_counter=0; **//extracting header and request lines**

while ((line\_header = b\_r.readLine()).length() != 0)

{

System.out.println(line\_header);

line\_counter++;

if(line\_counter==1)

line\_request=line\_header;

}

StringTokenizer Tokens = new StringTokenizer(line\_request); **//file\_name extractiion performed**

Tokens.nextToken();

String file\_name = Tokens.nextToken();

Path current\_relative\_path = Paths.get(""); **//file to be found to be in the same directory as the server file**

String s = current\_relative\_path.toAbsolutePath().toString(); **//setting the absolute path for the requested file and extracting the same file**

file\_name = s + file\_name;

System.out.println(file\_name); **//print the whole address of the file to be transmitted over the network**

FileInputStream f\_i\_s = null; **//client's requested file to be opened**

boolean file\_exists = true;

try

{

f\_i\_s = new FileInputStream(file\_name);

}

catch (FileNotFoundException error)

{

System.out.println(error);

file\_exists = false;

}

**//Response message to be displayed**

String line\_status = ""; **//displaying the current status**

String type\_line\_format = ""; **//displaying the format or type of line**

String body\_entity = ""; **//displaying the body entity**

**//connection paramters to be printed at the client side**

if (file\_exists) **//when true**

{

line\_status = "HTTP/1.1 200 OK" + CRLF;

type\_line\_format = "format-type: "+ format\_type(file\_name) + CRLF;

}

else **// when false**

{

line\_status ="HTTP/1.1 404 NOT FOUND"+CRLF ;

type\_line\_format = "format-type: NOT FOUND" + CRLF;

}

os.writeBytes(line\_status); **//displaying line status**

os.writeBytes(type\_line\_format); **//displaying format type**

InetAddress socketInetAddress = socket.getInetAddress();

String host\_Name = socketInetAddress.getHostName();

**//connection paramters to be printed at the client side**

os.writeBytes("Connection: Closed" + CRLF + "IP Address: " + host\_Name + CRLF); **//connection status**

String IP\_Type = " HTTP OVER TCP"; **//protocol used**

os.writeBytes("Protocol: "+IP\_Type +CRLF); **//displayed**

os.writeBytes("peer name: Server" + ":" + port\_numb + CRLF); **//peer name for client**

String socket\_Type = "Connection"; **//name for socket type**

os.writeBytes("Socket Type: " +socket\_Type +CRLF); **//displayed**

String socket\_Family = "AF\_INET"; **//name for socket family**

os.writeBytes("Socket Family: "+socket\_Family +CRLF); **//displayed**

os.writeBytes(CRLF); **//ending header line with a carriage return and line feed**

if (file\_exists) **//body content displayed**

{

bytes\_send(f\_i\_s, os);

f\_i\_s.close();

}

else

{

os.writeBytes("<HTML><TITLE>404 NOT FOUND</TITLE><BODY>404 FILE NOT FOUND - UNABLE TO FIND FILE</BODY></HTML>");

}

**//putting try/catch blocks around ports closing code**

try

{

os.close();

b\_r.close();

socket.close();

}

catch(Exception e)

{

System.out.println("error closing port");

}

System.out.println("---------------------------------------------------------------------------------");

}

}

}

**//server code ends**

***Client- Side Code***

**//import all the built-in classes to be used in the program from java**

import java.net.\* ; **//Provides for system input and output through data streams, serialization and the file system.**

import java.io.\* ; **//Provides the classes for implementing networking applications.**

**//defining client class and initializing the variables**

public class Client

{

final static String CRLF = "\r\n"; **//CRLF is carriage return line feed used for getting data from the format accepted by server**

public static void main(String[] argv) throws Exception{

**//IP address,port number and file\_name format in the command given to server;either via cmd or browser**

String file\_name = argv[2]; **//third keyword in the command in the cmd line or browser represents the file to look for in the server**

String IP\_address = argv[0]; **//initialization of ip address and port number**

int port\_numb = Integer.parseInt( argv[1] ); **// second keyword in the command in cmd line or browser represents the port to be used//changed to integer type here**

if(argv.length < 2)**//when the port number and the filename are not passed**

{

port\_numb = 8080;  **//set the port number to be 8080**

file\_name = "index.html"; **//set the default file to be index.html**

}

else

{

port\_numb = Integer.parseInt(argv[1]);**//else take whatever is passed for port number**

file\_name = argv[2]; **//else take whatever is passed for file name**

}

Socket Socket\_client = new Socket(IP\_address, port\_numb); **//creating socket for the client to communicate with the server**

PrintWriter os = new PrintWriter(Socket\_client.getOutputStream()); **//for the output bytes flowing from the client**

String response\_buffer = ""; **// buffer created for receiving data from the server response**

BufferedReader in = new BufferedReader(new InputStreamReader(Socket\_client.getInputStream())); **//for the incoming bytes to the client**

long start\_time = System.currentTimeMillis(); **//start time to calculate RTT**

long end\_time\_connection = System.currentTimeMillis(); **//calc the end time for request**

**//get the info from the client to the server (connection parameters for the server)**

os.print("REQUEST\_FROM\_CLIENT:" + "GET /" + file\_name + " HTTP/1.1" + CRLF + "Accept: text/html/htm,jpeg/jpg,gif" + CRLF + "Host Name:" + IP\_address + ":" + port\_numb + CRLF + "User-Agent:Command Prompt" + CRLF + "Accept-Encoding: UTF-8/Unicode" + CRLF + "Accept-Language: US eng" + CRLF + "Connection: keep-alive" + CRLF + "END TIME:" + end\_time\_connection + "ms" + CRLF + "peer name:" + IP\_address + port\_numb + CRLF +CRLF);

os.flush(); **//Flush the buffer**

**// RTT for the request/response cycle**

long end\_time = System.currentTimeMillis(); **//end time for RTT**

long total\_time = end\_time - start\_time; **//calc the actual RTT for the cycle**

**// printing the data from the response buffer**

while( (response\_buffer = in.readLine()) != null)

{

System.out.println(response\_buffer);

}

System.out.println("\nThe RTT for the request/response cycle is: " + total\_time + "ms");

}

}

**//client program ends**