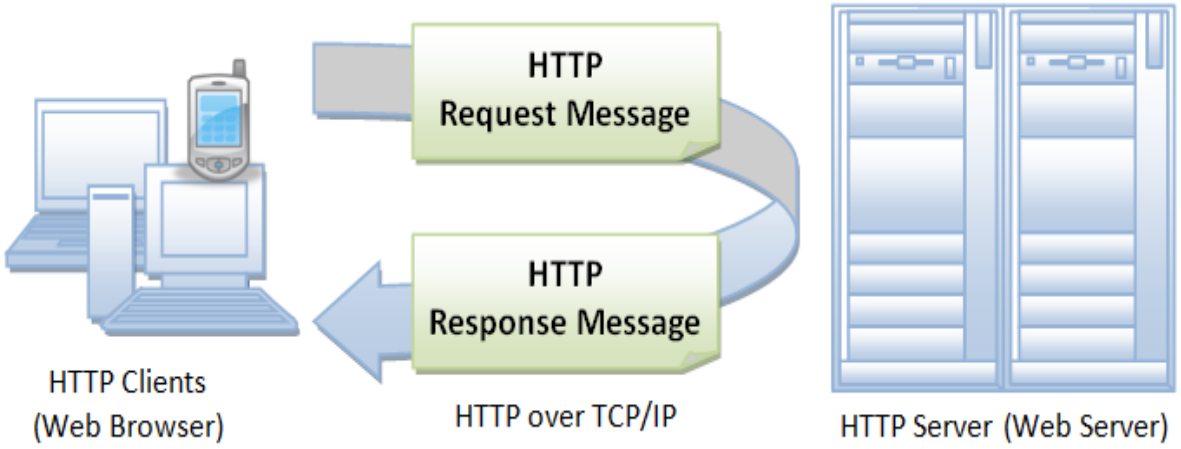
PROGRAMMING PROJECT 1

HTTP Client and Server

Rahul Ratra(801100033)
Shilpa Goel(801104237)

Introduction

HTTP: The Hypertext Transport Protocol (HTTP) is an application layer protocol that is used to transmit virtually all files and other data on the World Wide Web, whether they're HTML files, image files, query results, or anything else. Usually, HTTP takes place through TCP/IP sockets.
A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
A browser is an HTTP client because it sends requests to an HTTP server (Web server), which then sends responses back to the client. The standard (and default) port for HTTP servers to listen on is 80, though they can use any port.
HTTP is based on the TCP/IP protocols, and is used commonly on the Internet for transmitting web-pages from servers to browsers.
HTTP is an asymmetric request-response client-server protocol as illustrated below. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message.



HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.
HTTP permits negotiating of data type and representation, so as to allow systems to be built independently of the data being transferred.

HTTP Client: An HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a server (typically port 80, occasionally port 8080; see List of TCP and UDP port numbers).

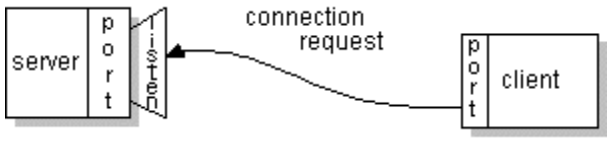
HTTP Server: An HTTP server listening on that port waits for a client's request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of this message is typically the requested resource, although an error message or other information may also be returned.
There are request methods to send the request which are as follows:

GET: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
POST: The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI.
PUT: The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI.
DELETE: The DELETE method deletes the specified resource.

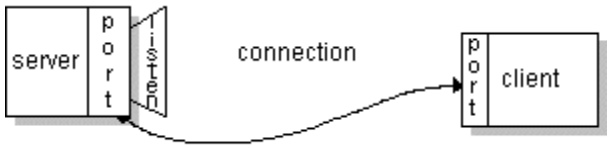
Client Server Communication using Sockets

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.

On the client-side: The client knows the host name 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.



On the server-side: If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server.

The client and server can now communicate by writing to or reading from their sockets.

Project Implementation Details

The objective of this project is to implement a HTTP Client and HTTP Server that runs a simplified version of HTTP/1.1 .This program will implement two commands :GET and PUT.

We have used Java Programming Language to implement it. There are two projects : HTTP Client and HTTP Server .
Details for both of them are:

HTTP Client : In this, client side implementation has been done i.e Client is taking the following command line arguments (in order): server name, port on which to contact the server, HTTP command (GET or PUT), and the path of the requested object on the server.

There are two main files:

- **MyClient.java:** It accepts the command line arguments and sends the flow to MyClientImpl.java for further processing. It has the main method() to start the flow.
- **MyClientImpl.java :** This is the implementation file at client for handling the GET , PUT and Terminate signal request. The methods defined below are used to handle these requests:
  getMethod(String, int, String)
  putMethod(String, int, String)
  readFile(String)
  terminateServerSignal(String, int, int)

In response to a GET command, the client is:

a. Getting connected to the server via a TCP connection
b. Able to submit a valid HTTP/1.1 GET request to the server
c. Able to read the server's response and display it

In response to a PUT command,the client is:

a. Getting connected to the server via aTCP connection
b. Able to submit a valid HTTP/1.1 PUT request to the server
c. Able to send the file to the server
d. Read the server's response and display it

HTTP Server: In this, server side implementation has been done i.e Server is taking port number in the command line argument. There are three main files:

- **MyServer.java:** It accepts the port number as the command line argument and sends the flow to MultiThreadedServer.java for further processing. It has the main method() to start the flow.
- **MultiThreadedServer.java:** This is to handle multiple requests from the client.
- **SocketClientHandler.java :** This is the implementation file at server side for handling the GET , PUT and Terminate signal request from client. The methods defined below are used to handle these requests:
  checkURL(String)
  constructResponseHeader(int, StringBuilder)
  getDatagetData(String)
  getTimeStamp()
  putData(String)
  returnSignalValue()

writeHtmlFile(String)

e. readResponse()

Execution Details:

1. **Start Server:** Go to Command Prompt and enter the below command with port number on which you want to listen client requests.(Assuming code is already compiled):

Sample Input: java MyServer 8090

```
==========================================
rahuls-MacBook-Pro:src rahulratra$ java MyServer 8090
Starting the socket server at port:8090
Waiting for clients...
```

2. **For GET request:** Go to command prompt at client and enter the command line arguments having the server name, port on which to contact the server, HTTP command GET , and the path of the requested object on the server. If the file exists, user will get the 200 response and 404 in case file doesn't exist.

Sample input: java MyClient localhost 8090 GET /Users/rahulratra/git/SpringDemo/HTTPClient/src/Test2.txt

If file exist:

```
rahuls-MacBook-Pro:src rahulratra$ java MyClient localhost 8090 GET /Users/rahulratra/git/SpringDemo/HTTPClient/src/Test1.txt
======================================
Connected
======================================
Request Sent!
======================================
HTTP/1.1 200 OK
Date:03/10/2019 7:06:58 PM
Server:localhost
Content-Type: text/html
Connection: Closed

Watching that frenzy of insects above the bush of white flowers,
bush I see everywhere on hill after hill, all I can think of
is how terrifying spring is, in its tireless, mindless replications.
Everywhere emergence: seed case, chrysalis, uterus, endless manufacturing.
And the wrapped stacks of Styrofoam cups in the grocery, lately
I can't stand them, the shelves of canned beans and soups, freezers
of identical dinners; then the snowflake-diamond-snowflake of the rug
beneath my chair, rows of books turning their backs,
even my two feet, how they mirror each other oppresses me,
the way they fit so perfectly together, how I can nestle one big toe into the other
like little continents that have drifted; my God the unity of everything,
my hands and eyes, yours; doesn't that frighten you sometimes, remembering
the pleasure of nakedness in fresh sheets, all the lovers there before you,
beside you, crowding you out? And the scouring griefs,
don't look at them all or they'll kill you, you can barely encompass your own;
I'm saying I know all about you, whoever you are, it's spring
and it's starting again, the longing that begins, and begins, and begins.
======================================
Response Recieved!!
======================================
rahuls-MacBook-Pro:src rahulratra$
```

If file doesn't exist:

```
--------------------------------------
rahuls-MacBook-Pro:src rahulratra$ java MyClient localhost 8090 GET /Users/rahulratra/git/SpringDemo/HTTPClient/src/Test2.txt
======================================
Connected
======================================
Request Sent!
======================================
HTTP/1.1 404 Not Found
Date:03/10/2019 7:07:25 PM
Server:localhost

======================================
Response Recieved!!
======================================
rahuls-MacBook-Pro:src rahulratra$
```

3. **For Put request:** It will put data on server and return the response 200 in case of success and 404 in case of error.

   Sample input: java MyClient localhost 8090 PUT /Users/rahulratra/git/SpringDemo/HTTPClient/src/Test2.txt

```
Last login: Sun Mar 10 18:47:04 on console
rahuls-MacBook-Pro:~ rahulratra$ cd /Users/rahulratra/git/SpringDemo/HTTPClient/src/
rahuls-MacBook-Pro:src rahulratra$ javac MyCLient.java
rahuls-MacBook-Pro:src rahulratra$ java MyClient localhost 8090 PUT /Users/rahulratra/git/SpringDemo/HTTPClient/src/Test1.txt
====================================
Connected
====================================
PUT Request Header Sent!
====================================
html content:Watching that frenzy of insects above the bush of white flowers,
bush I see everywhere on hill after hill, all I can think of
is how terrifying spring is, in its tireless, mindless replications.
Everywhere emergence: seed case, chrysalis, uterus, endless manufacturing.
And the wrapped stacks of Styrofoam cups in the grocery, lately
I can't stand them, the shelves of canned beans and soups, freezers
of identical dinners; then the snowflake-diamond-snowflake of the rug
beneath my chair, rows of books turning their backs,
even my two feet, how they mirror each other oppresses me,
the way they fit so perfectly together, how I can nestle one big toe into the other
like little continents that have drifted; my God the unity of everything,
my hands and eyes, yours; doesn't that frighten you sometimes, remembering
the pleasure of nakedness in fresh sheets, all the lovers there before you,
beside you, crowding you out? And the scouring griefs,
don't look at them all or they'll kill you, you can barely encompass your own;
I'm saying I know all about you, whoever you are, it's spring
and it's starting again, the longing that begins, and begins, and begins.

PUT Data Sent!
====================================
HTTP/1.1 200 OK
Date:03/10/2019 7:05:55 PM
Server:localhost
Content-Type: text/html
Connection: Closed

====================================
Response Recieved!!
====================================
```

4. **To shut down server:** Client will send the termination signal request and then server will shut down

   Sample input: java MyClient localhost 8090 66

```
rahuls-MacBook-Pro:src rahulratra$ java MyClient localhost 8090 66
====================================
Connected
====================================
Request Sent!
====================================
rahuls-MacBook-Pro:src rahulratra$
```

```
Inside readResponse() method
Value of temp is:66
Updated value of terminateSignal is:true
SocketClientHandler.returnSignalValue() is:true
Out of the while loop
========================================
Server is shut down now!!
========================================
rahuls-MacBook-Pro:src rahulratra$
```

Source Code:

Rahul Ratra and Shilpa Goel

1. MyClient.java

```
import java.io.IOException;

public class MyClient {

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

		String host = "";
		int port = 0;
		String command = "";
		String path = "";
		int signal = 0;
		// fetch the command line arguments entered by user
		if (args.length == 4) {
			host = args[0];
			port = Integer.parseInt(args[1]);
			command = args[2];
			path = args[3];
		} else if (args.length == 3 && args[2].equalsIgnoreCase("66")) {
			host = args[0];
			port = Integer.parseInt(args[1]);
			signal = Integer.parseInt(args[2]);
		} else {
			System.out
					.println("Invalid Arguments!!! Please enter
valid arguments.");
			return;
		}

		// Method Check GET or PUT or Terminal Signal
		if ("GET".equals(command)) {
			MyClientImpl.getMethod(host, port, path);
		} else if ("PUT".equals(command)) {
			MyClientImpl.putMethod(host, port, path);
		} else if (signal == 66) {
			MyClientImpl.terminateServerSignal(host, port, signal);
		} else {
			System.out
					.println("Check the HTTP command! It should be
either GET or PUT");
			return;
		}
	}

}
```

2. MyClientImpl.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

public class MyClientImpl {

	// this method handles get request from the client
	public static void getMethod(String host, int port, String path)
			throws IOException {

		// Opening Connection based on the port number 80(HTTP)
		Socket clientSocket = null;
		clientSocket = new Socket(host, port);

		System.out.println("=====================================");
		System.out.println("Connected");
		System.out.println("=====================================");

		// Declare a writer to this url
		PrintWriter request = new
PrintWriter(clientSocket.getOutputStream(),
				true);

		// Declare a listener to this url
		BufferedReader response = new BufferedReader(new
InputStreamReader(
				clientSocket.getInputStream()));

		// Sending request to the server
		// Building HTTP request header
		request.print("GET /" + path + "/ HTTP/1.1\r\n"); // "+path+"
		request.print("Host: " + host + "\r\n");
		request.print("Connection: close\r\n");
		request.print("Mozilla/4.0 (compatible; MSIE5.01; Windows
NT)\r\n");
		request.print("\r\n");
		request.flush();
		System.out.println("Request Sent!");
		System.out.println("=====================================");

		// Receiving response from server
		String responseLine;
		while ((responseLine = response.readLine()) != null) {
			System.out.println(responseLine);
		}
		System.out.println("=====================================");
		System.out.println("Response Recieved!!");
		System.out.println("=====================================");

		response.close();
		request.close();
		clientSocket.close();
	}

	// this method handles put request from the client
	public static void putMethod(String host, int port, String file)
			throws UnknownHostException, IOException {
```

```
            // Opening Connection based on the port number 80(HTTP)
            Socket clientSocket = null;
            clientSocket = new Socket(host, port);

            System.out.println("====================================");
            System.out.println("Connected");
            System.out.println("====================================");

            PrintWriter request = new
PrintWriter(clientSocket.getOutputStream(),
                        true);
            BufferedReader response = new BufferedReader(new
InputStreamReader(
                        clientSocket.getInputStream()));

            // Sending request to the server
            // Building HTTP request header
            request.print("PUT /" + file + "/ HTTP/1.1\r\n"); // "+path+"
            request.print("Host: " + host+"\r\n");
            request.print("Accept-Language: en-us\r\n");
            request.print("Connection: Keep-Alive\r\n");
            request.print("Mozilla/4.0 (compatible; MSIE5.01; Windows
NT)\r\n");
            request.print("Content-type: text/html\r\n");
            request.print("Content-Length: 0\r\n");
            request.print("\r\n");

            System.out.println("PUT Request Header Sent!");
            System.out.println("====================================");

            // Send the Data to be PUT
            String htmlContent = readFile(file);
            System.out.println("html content:" + htmlContent);
            request.println(htmlContent);
            request.flush();

            System.out.println("PUT Data Sent!");
            System.out.println("====================================");

            // Receiving response from server
            String responseLine;
            while ((responseLine = response.readLine()) != null) {
                  System.out.println(responseLine);
            }
            System.out.println("====================================");
            System.out.println("Response Recieved!!");
            System.out.println("====================================");
            request.close();
            response.close();
            clientSocket.close();
      }

      // this method handles terminate signal request
      public static void terminateServerSignal(String host, int port, int
signal)
                  throws UnknownHostException, IOException {
```

```
            Socket clientSocket = null;
            clientSocket = new Socket(host, port);
            System.out.println("====================================");
            System.out.println("Connected");
            System.out.println("====================================");
            PrintWriter request = new
PrintWriter(clientSocket.getOutputStream(),
                        true);
            request.print(signal);
            request.flush();
            System.out.println("Request Sent!");
            System.out.println("====================================");
            request.close();
            clientSocket.close();
      }

      // this method reads the file content to sent to the server
      private static String readFile(String file) {

            StringBuilder sb = new StringBuilder();
            try {
                  BufferedReader br = new BufferedReader(new
FileReader(file));

                  String line = br.readLine();
                  while (line != null) {
                        sb.append(line).append("\n");
                        line = br.readLine();
                  }

                  String fileAsString = sb.toString();

            } catch (FileNotFoundException e) {

                  System.err.println(e.getMessage());
                  System.out.println("Client side issue with file name");
                  System.exit(1);
                  // stringBuilder.setLength(0);
                  // stringBuilder.append("Check the file name!");
                  // return stringBuilder.toString();

            } catch (IOException e) {

                  System.err.println(e.getMessage());
                  System.exit(1);
                  // stringBuilder.setLength(0);
                  // stringBuilder.append("File has no text!");
                  // return stringBuilder.toString();
            }
            return sb.toString();
      }
}
```

3. MyServer.java

```
import java.io.IOException;
```

```
class MyServer {

	public static void main(String[] args) throws InterruptedException {
		int portNumber = Integer.parseInt(args[0]);

		try {
			// initializing the Socket Server
			MultiThreadedServer socketServer = new MultiThreadedServer(
					portNumber);
			socketServer.start();

		} catch (IOException e) {
			e.printStackTrace();
		}
	}

}
```

4. SocketClientHandler.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;

public class SocketClientHandler implements Runnable {

	private Socket client;
	private int i = 0;
	public static boolean terminateSignal = false;

	public SocketClientHandler(Socket client) {
		this.client = client;
	}

	@Override
	public void run() {
		try {
			System.out.println("Thread started with name is :"
					+ Thread.currentThread().getName());
			readResponse();
			return;
		} catch (IOException e) {
			e.printStackTrace();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
```

```
	public static boolean returnSignalValue() {
		return terminateSignal;
	}

	private void readResponse() throws IOException, InterruptedException {

		try {

			System.out.println("-----");
			BufferedReader request = new BufferedReader(new
InputStreamReader(
					client.getInputStream()));
			BufferedWriter response = new BufferedWriter(
					new
OutputStreamWriter(client.getOutputStream()));

			System.out.println("Inside readResponse() method");
			String putDataFromClient = "";
			String requestHeader = "";
			String temp = ".";
			while (!temp.equals("")) {
				temp = request.readLine();
				System.out.println("Value of temp is:"+temp);
				if (temp.equals("66")) {
					requestHeader = temp;
					request.close();
					client.close();
					terminateSignal = true;
					System.out.println("Updated value of
terminateSignal is:" + terminateSignal);
					return;

				}
				requestHeader += temp + "\n";
			}
			System.out.println("RequestHeader from client is:" +
requestHeader);

			// Get the method from HTTP header
			StringBuilder sb = new StringBuilder();
			String file = requestHeader.substring(
					requestHeader.indexOf("/") + 1,
					requestHeader.indexOf(".txt")).trim();
			System.out.println("file:" + file);
			System.out.println("line no 61");
			System.out.println("=============");
			System.out.println("header123:" + requestHeader);
			System.out.println("=============");
			System.out.println("true or false:"
					+
requestHeader.split("\n")[0].contains("PUT"));
			System.out.println("line no 67");
			if (requestHeader.split("\n")[0].contains("GET")
					&& checkURL(file + ".txt")) {

				System.out.println("Hi Inside line no 73");
```

```
                        // Get the correct page
                        constructResponseHeader(200, sb);
                        response.write(sb.toString());
                        response.write(getData(file + ".txt"));
                        sb.setLength(0);
                        response.flush();
                  } else if (requestHeader.split("\n")[0].contains("PUT")) {

                        System.out.println("line n0 75");
                        System.out.println("hi inside lino number 70");
                        // Get the data from the inputStream

                        StringBuilder sb1 = new StringBuilder();
                        putDataFromClient = request.readLine();
                        System.out.println("putdatafromclient:"
                                    + putDataFromClient.length());
                        while (putDataFromClient != null
                                    && putDataFromClient.length() != 0) {
                              System.out.println("line no 87:" +
putDataFromClient);
                              sb1.append(putDataFromClient).append("\n");
                              putDataFromClient = request.readLine().trim();
                        }

                        String dataStream = sb1.toString();
                        System.out.println("data from client:" +
sb1.toString());

                        // PUT the data to file serverIndex.html
                        if (dataStream != "") {
                              System.out.println("line no 96");
                              int responseCode = putData(dataStream);
                              constructResponseHeader(responseCode, sb);
                              response.write(sb.toString());
                              sb.setLength(0);
                              response.flush();
                        } else {
                              constructResponseHeader(304, sb);
                              response.write(sb.toString());
                              sb.setLength(0);
                              response.flush();
                        }

                  } else {
                        // Enter the error code
                        // 404 page not found
                        constructResponseHeader(404, sb);
                        response.write(sb.toString());
                        sb.setLength(0);
                        response.flush();
                  }

                  request.close();
                  response.close();

                  client.close();
                  return;
```

```
            } catch (Exception e) {
                  System.out.println("exception trace:::" + e.getMessage());
            }

      }

      // Check the URL from the Request header to the server's database
      private static boolean checkURL(String file) {

            System.out.println("file:" + file);
            File myFile = new File(file);
            // System.out.println(file);
            // System.out.println("IT IS CHEKCING");
            // System.out.println(myFile.exists() && !myFile.isDirectory());
            return myFile.exists() && !myFile.isDirectory();

      }

      // Construct Response Header
      public static void constructResponseHeader(int responseCode,
                  StringBuilder sb) {

            if (responseCode == 200) {

                  sb.append("HTTP/1.1 200 OK\r\n");
                  sb.append("Date:" + getTimeStamp() + "\r\n");
                  sb.append("Server:localhost\r\n");
                  sb.append("Content-Type: text/html\r\n");
                  sb.append("Connection: Closed\r\n\r\n");

            } else if (responseCode == 404) {

                  sb.append("HTTP/1.1 404 Not Found\r\n");
                  sb.append("Date:" + getTimeStamp() + "\r\n");
                  sb.append("Server:localhost\r\n");
                  sb.append("\r\n");
            } else if (responseCode == 304) {
                  sb.append("HTTP/1.1 304 Not Modified\r\n");
                  sb.append("Date:" + getTimeStamp() + "\r\n");
                  sb.append("Server:localhost\r\n");
                  sb.append("\r\n");
            } else if (responseCode == 201) {

                  sb.append("HTTP/1.1 200 OK\r\n");
                  sb.append("Date:" + getTimeStamp() + "\r\n");
                  sb.append("Server:localhost\r\n");
                  sb.append("Content-Type: text/html\r\n");
                  sb.append("Server Connection: now Closed\r\n\r\n");
            }

      }

      // PUT data to file ServerIndex.htm
      private static int putData(String putDataFromClient) throws
IOException {

            return writeHtmlFile(putDataFromClient);
```

```
	}

	private static String getData(String file) {

		StringBuilder sb = new StringBuilder();
		try {
			BufferedReader br = new BufferedReader(new
FileReader(file));

			String line = br.readLine();
			while (line != null) {
				sb.append(line).append("\n");
				line = br.readLine();
			}

			String fileAsString = sb.toString();

		} catch (FileNotFoundException e) {

			System.err.println(e.getMessage());
			System.out.println("Client side issue with file name");
			System.exit(1);
			// stringBuilder.setLength(0);
			// stringBuilder.append("Check the file name!");
			// return stringBuilder.toString();

		} catch (IOException e) {

			System.err.println(e.getMessage());
			System.exit(1);
			// stringBuilder.setLength(0);
			// stringBuilder.append("File has no text!");
			// return stringBuilder.toString();
		}
		return sb.toString();
	}

	// Write the data to server - Helper method for putData method
	private static int writeHtmlFile(String putDataFromClient) {

		System.out.println("line 204:" + putDataFromClient);
		System.out.println("location of the file:");
		File myFile = new File(
				"/Users/rahulratra/git/SpringDemo/HTTPServer/src/"
					+ System.currentTimeMillis() + ".txt");
		BufferedWriter writer;
		try {
			writer = new BufferedWriter(new FileWriter(myFile));
			writer.write(putDataFromClient);
			writer.close();
			return 200;
		} catch (IOException e) {
			return 304;
		}
	}

	// TimeStamp
```

```
	private static String getTimeStamp() {
		Date date = new Date();
		SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy h:mm:ss
a");
		String formattedDate = sdf.format(date);
		return formattedDate;
	}

}
```

5. MultiThreadedSer ver.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class MultiThreadedServer {

	private ServerSocket serverSocket;
	private int port;

	public MultiThreadedServer(int port) {
		this.port = port;
	}

	public void start() throws IOException, InterruptedException {

		serverSocket = new ServerSocket(port);
		System.out.println("Starting the socket server at port:" +
port);

		Socket client = null;

		while (true) {

			System.out.println("Waiting for clients...");
			client = serverSocket.accept();
			System.out
					.println("The following client has connected to
the server:"
							+
client.getInetAddress().getCanonicalHostName());
			Thread t = Thread.currentThread();

			System.out.println("Current thread: " + t.getName());
			// A client has connected to this server
			Thread thread = new Thread(new
SocketClientHandler(client));
			thread.start();
			try {
				System.out.println("Current Thread: "
						+ Thread.currentThread().getName());
```

```
                        thread.join();
                  }

                  catch (Exception ex) {
                        System.out.println("Exception has been caught" + ex);
                  }
                  System.out.println("SocketClientHandler.returnSignalValue()
is:"
                              + SocketClientHandler.returnSignalValue());
                  if (SocketClientHandler.returnSignalValue()) {
                        break;
                  }
            }
            System.out.println("Out of the while loop");
            serverSocket.close();
            System.out.println("=======================================");
            System.out.println("Server is shut down now!!");
            System.out.println("=======================================");
      }

}
```