Session 1

OBJECTIVE: In this lab students will learn the basics of Java Socket programming and create comprehensive server-client application using TCP socket.

Problem 1

Problem Description: Write an application program to implement TCP iterative Server-Client application to convert the given input sentence to uppercase. In this program user will enter a sentence and client will send the sentence to the server through client socket. Server read the sentence send by client through the server socket. Server converts the sentence to upper case and sends the modified sentence to client. Client display the modified sentence in the input consol.

Implementation steps for Server Application:

1. CREATE a class of name TCPServer.
2. CREATE server socket at any port to listen the client.
   ServerSocket welcomeSocket = new ServerSocket(6789);
3. LISTENS for new connection and when a connection arrives, accept it.
   Socket connectionSocket = welcomeSocket.accept();
4. READ LINE from socket. Get the input stream attached with the socket calling getInputStream() and wrap it with text based wrapper BufferedReader.
   BufferedReader inFromClient = new BufferedReader (new InputStreamReader (connectionSocket.getInputStream()));
5. Get the output stream attached with the socket calling getOutputStream().
   DataOutputStream outToClient = new DataOutputStream (connectionSocket.getOutputStream());
6. READ in line from socket:
   String clientSentence = inFromClient.readLine();
7. CONVERT the line to uppercase:
   String capitalizedSentence = clientSentence.toUpperCase() + '\n';
8. WRITE out line to socket:
   outToClient.writeBytes(capitalizedSentence);
9. CLOSE the connection.
   welcomeSocket.close();

Implementation steps for Client Application:

1. CREATE a class of name TCPClient.
2. CREATE client socket and connect it to the server's port number and local host.
   Socket clientSocket = new Socket("localhost", 6789);

3. READ LINE by creating input stream attached to standard input:

   BufferedReader inFromUser =

   new BufferedReader(new InputStreamReader(System.in));

4. READ from user

   String sentence = inFromUser.readLine();

5. SEND it to a server via client socket.

   DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

   BufferedReader inFromServer = new BufferedReader(new

   InputStreamReader(clientSocket.getInputStream()));

6. READ server's response from socket and displays the modified line.

   outToServer.writeBytes(sentence + '\n');

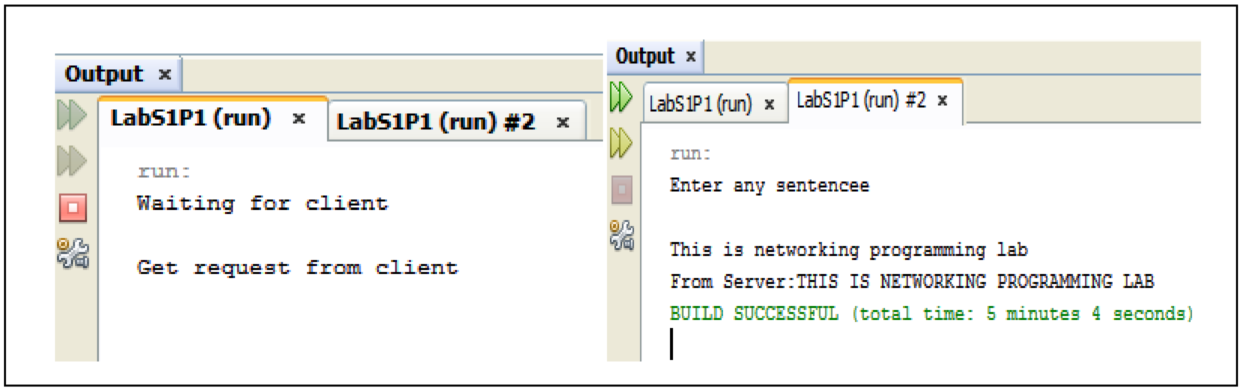
   String modifiedSentence = inFromServer.readLine();

   System.out.println("FROM SERVER: " + modifiedSentence);

7. CLOSE the client socket.

   clientSocket.close();

Sample Input/output:



```
run:
Waiting for client

Get request from client
```

```
run:
Enter any sentencee

This is networking programming lab
From Server:THIS IS NETWORKING PROGRAMMING LAB
BUILD SUCCESSFUL (total time: 5 minutes 4 seconds)
```

Lab Task 01:

Problem Description:

i. Develop a "Remote Calculator" application that works as follows:
   - The client program inputs two integers and an arithmetic operation ('*','/','%','+','-') from the user and sends these three values to the server side using a simple protocol ***operator:first-operand:second-operand***.

- The server does the binary operation on the two integers and sends backs the result of the operation to the client.
- The client displays the result to the user.

ii. Extend the functionalities of the server/client application of (i) as follows:

- The client program can now also input one integer and a unary operation (sqrt(), sin(), cos()) from a user and sends these two values to server side using a similar protocol ***operator:operand.***
- The server does the binary or unary operation depending on the message received from the client.
- The server also checks for the invalid operation and/or operand and sends corresponding message.
- The server also checks for math error (e.g, divide by zero error for division operation and negative operand for sqrt() operation) and sends corresponding error message.

Offline Exercises:

1. Develop a simple chatting application using (i) Connection-oriented and (ii) Connectionless sockets. In each case, when the user presses the "Enter" key, whatever characters have been typed by the user until then are transferred to the other end. You can also assume that for every message entered from one end, a reply must come from the other end, before another message could be sent. In other words, more than one message cannot be sent from a side before receiving a response from the other side. For connectionless communication, assume the maximum number of characters that can be transferred in a message to be 1000. The chat will be stopped by pressing Ctrl+C on both sides.

2. Implement a simple file transfer protocol (FTP) using connection-oriented. The connection-oriented FTP works as follows: At the client side, the file to be transferred is divided into units of 100 bytes (and may be less than 100 bytes for the last unit depending on the size of the file). The client transfers each unit of the file to the server and expects an acknowledgment from the server. Only after receiving an acknowledgment from the server, the client transmits the next unit of the file. If the acknowledgment is not received within a timeout period (choose your own value depending on your network delay), the client retransmits the unit. The above process is repeated until all the contents of the file are transferred.