ONE WAY COOMMUNICATION

Problem Statement - Develop a basic TCP-based client-server communication system in Java using sockets. In this setup, the server listens for incoming connections on port 6666 using a ServerSocket, accepts a connection from a client, receives a UTF-encoded message sent by the client through a DataInputStream, and displays the message on the server console. On the other side, the client connects to the server using the localhost address and the same port number, sends a UTF-encoded message ("Hello Server") using a DataOutputStream, and then closes the connection. This program demonstrates simple message transfer over a TCP/IP network using Java's networking and I/O libraries.

Algorithm:

For Server Program –

Step 1: Start the program.

Step 2: Create a ServerSocket object on port 6666 to listen for incoming client connections.

Step 3: Wait and accept a client connection using the accept() method, which returns a Socket object when a connection is established.

Step 4: Create a DataInputStream object to receive input from the client through the input stream of the connected socket.

Step 5: Read a UTF-encoded message from the client using readUTF() and store it in a string variable.

Step 6: Display the received message on the server console.

Step 7: Close the ServerSocket to release the port and end the server process.

Step 8: End the program.

For Client Program –

Step 1: Start the program.

Step 2: Create a Socket object to connect to the server at address "localhost" and port 6666.

Step 3: Create a DataOutputStream object to send data through the output stream of the socket.

Step 4: Write a UTF-encoded message (e.g., "Hello Server") to the server using writeUTF().

Step 5: Flush the output stream to ensure all data is sent using flush().

Step 6: Close the DataOutputStream to stop writing data.

Step 7: Close the Socket to end the connection with the server.

Step 8: End the program.

Implementation:

For Server Program –

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

For Client Program –

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

STOP & WAIT ARQ

Problem Description: Develop a bi-directional message exchange system between a client and a server using TCP sockets in Java. The server runs on port 8080, waits for a client connection, and once connected, both the server and the client can exchange messages in a loop until either side sends the termination command "stop". Upon receiving a message, the server acknowledges the message by sending an "ACK" back to the client and then waits for the user to input a response message to send to the client. Similarly, the client sends a message entered by the user, receives the acknowledgment from the server, and continues the cycle. This interactive communication demonstrates the use of ServerSocket, Socket, DataInputStream, and DataOutputStream, and illustrates how two-way communication can be established and maintained over a TCP/IP connection using Java networking.

Algorithm:

For Server Program –

Step 1: Start the program.

Step 2: Create a ServerSocket object to listen on port 8080.

Step 3: Display a message indicating that the server is listening.

Step 4: Wait for a client connection using accept(), and store the resulting Socket object.

Step 5: Display a message confirming that the client is connected.

Step 6: Create DataInputStream and DataOutputStream objects using the socket's input and output streams.

Step 7: Repeat the following steps until either the sent or received message is "stop" (case-insensitive):

- Read a UTF-encoded message from the client using readUTF().
- Display the received message on the server console.
- Send an acknowledgment message "ACK" back to the client using writeUTF() and flush().
- Prompt the server user to enter a message via the console.
- Read the message from the user and send it to the client using writeUTF() and flush().
- Display the sent message on the server console.

Step 8: After exiting the loop, close DataInputStream, DataOutputStream, the Socket, and the ServerSocket.

Step 9: End the program.

For Client Program –

Step 1: Start the program.

Step 2: Create a Socket object to connect to the server running on "localhost" at port 8080.

Step 3: Display a message indicating that the client is connected to the server.

Step 4: Create DataInputStream and DataOutputStream objects using the socket's input and output streams.

Step 5: Repeat the following steps until either the sent or received message is "stop" (case-insensitive):

- Prompt the user to enter a message via the console.
- Read the message from the user.
- Send the UTF-encoded message to the server using writeUTF() and flush().
- Display the sent message on the client console.
- Receive a UTF-encoded acknowledgment message from the server using readUTF().
- Display the received acknowledgment on the console.

Step 6: After exiting the loop, close the DataInputStream, DataOutputStream, and Socket.

Step 7: End the program.

Implementation:

For Server Program –

```
import java.io.*;

import java.net.*;

import java.util.*;

public class Serversnw {

private static ServerSocket serverSocket;

private static Socket socket;

private static DataInputStream dataInputStream;

private static DataOutputStream dataOutputStream;

public static void main(String[] args) {

try {

serverSocket = new ServerSocket(8080);

System.out.println("Server is listening on port 8080...");

socket = serverSocket.accept();

System.out.println("Client connected.");

dataInputStream = new DataInputStream(socket.getInputStream());

dataOutputStream = new DataOutputStream(socket.getOutputStream());
String receivedMessage;
String sentMessage;
do {
// Receive message from client
receivedMessage = dataInputStream.readUTF();
System.out.println("Received: " + receivedMessage);
// Send ACK to client
dataOutputStream.writeUTF("ACK");
dataOutputStream.flush();
System.out.println("Sent ACK.");
// Send message to client
System.out.print("Enter message to send: ");
sentMessage = new Scanner(System.in).nextLine();
dataOutputStream.writeUTF(sentMessage);
dataOutputStream.flush();
System.out.println("Sent: " + sentMessage);
} while (!sentMessage.equalsIgnoreCase("stop") && !receivedMessage.equalsIgnoreCase("stop"));
// Close resources
dataInputStream.close();
dataOutputStream.close();
socket.close();
serverSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

For Client Program –

```
import java.io.*;
import java.net.*;
import java.util.*;
```

```
public class Clientsnw {
private static Socket socket;
private static DataInputStream dataInputStream;
private static DataOutputStream dataOutputStream;
public static void main(String[] args) {
try {
socket = new Socket("localhost", 8080);
System.out.println("Connected to server.");
dataInputStream = new DataInputStream(socket.getInputStream());
dataOutputStream = new DataOutputStream(socket.getOutputStream());
String receivedMessage;
String sentMessage;
do {
// Send message to server
System.out.print("Enter message to send: ");
sentMessage = new Scanner(System.in).nextLine();
dataOutputStream.writeUTF(sentMessage);
dataOutputStream.flush();
System.out.println("Sent: " + sentMessage);
// Receive ACK from server
receivedMessage = dataInputStream.readUTF();
System.out.println("Received: " + receivedMessage);
} while (!sentMessage.equalsIgnoreCase("stop") && !receivedMessage.equalsIgnoreCase("stop"));
// Close resources
dataInputStream.close();
dataOutputStream.close();
socket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
```

NUMBERS AND ALPHABETS SEQUENCE

Problem Description: Develop a simple character-processing communication system between a client and a server using TCP sockets. The server listens for incoming connections on port 1234, and once a client connects, both sides exchange messages interactively. The communication continues until either party sends the termination keyword "Stop".

The program includes a minor transformation operation on the first character of the received string. On the server side, when a message is received from the client, the first character is incremented by one in the ASCII sequence and displayed. Then, the server waits for manual input from the user to send a response back. On the client side, it sends input strings to the server, receives the server's response, and again increments the first character before displaying it. This interactive and character-manipulating exchange highlights the use of Java's Socket, ServerSocket, DataInputStream, and DataOutputStream classes, along with console I/O handling using BufferedReader. The entire system demonstrates basic socket programming concepts, character operations, and bidirectional communication between a client and server.

Algorithm:

For Server Program –

```
Step 1:  Start the program.
Step 2: Create a ServerSocket on port 1234.
Step 3:  Wait and accept a client connection using accept(); store it in a Socket object.
Step 4: Initialize the input stream (DataInputStream) to receive data from the client.
Step 5: Initialize the output stream (DataOutputStream) to send data to the client.
Step 6:  Initialize a BufferedReader to read input from the server-side user.
Step 7:  Initialize two string variables str and str2 as empty strings.
Step 8: Repeat the following steps until str is equal to "Stop":
    • Read a string str from the client using readUTF().
    • If str equals "Stop", set str2 = str to prepare for graceful termination.
    • Else:
        o Extract the first character from str using charAt(0).
        o Increment the character by 1 (ASCII value).
        o Display the incremented character on the server console.
        o Read a line from the server user via BufferedReader and store it in str2.
    • Send str2 to the client using writeUTF() and flush the output.
Step 9:  After the loop ends, close the DataInputStream, Socket, and ServerSocket.
Step 10:  End the program.
```

<u>For Client Program –</u>

<u>Step 1:</u> Start the program.

<u>Step 2:</u> Create a Socket object to connect to the server on localhost at port 1234.

<u>Step 3:</u> Initialize DataInputStream to receive data from the server.

<u>Step 4:</u> Initialize DataOutputStream to send data to the server.

<u>Step 5:</u> Initialize a BufferedReader to read input from the client-side user.

<u>Step 6:</u> Initialize two string variables str and str2 as empty strings.

<u>Step 7:</u> Repeat the following steps until str is equal to "Stop":

- Read a string from the user using BufferedReader and store it in str.
- Send the string str to the server using writeUTF() and flush the stream.
- Read the response from the server using readUTF() and store it in str2.
- If str2 equals "Stop", break the loop.
- Otherwise:
    - Extract the first character from str2.
    - Increment the character by 1 (in ASCII).
    - Print the incremented character to the console.

<u>Step 8:</u> After exiting the loop, flush and close the DataOutputStream.

<u>Step 9:</u> Close the socket connection.

<u>Step 10:</u> End the program.

<u>Implementation:</u>

<u>For Server Program –</u>

```
import java.net.*;

import java.io.*;

public class Serverabcd{

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

ServerSocket ss = new ServerSocket(1234);

Socket s = ss.accept();

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str = "", str2 = "";
while(!str.equals("Stop")){
str = din.readUTF();
if(str.equals("Stop"))
{
str2 = str;
}
else
{
char ch = str.charAt(0);
ch+=1;
System.out.println(ch);
str2 = br.readLine();
}
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}
}
```

For Client Program –

```
import java.net.*;
import java.io.*;
public class Clientabcd{
public static void main(String arg[]) throws Exception{
//InetAddress ia = InetAddress.getLocalHost();
Socket s = new Socket("localhost",1234);
DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
```

NUMBERS AND ALPHABETS SEQUENCE IN REVERSE ORDER

Problem Description: Develop a simple client-server communication model using socket programming. The objective is to establish a two-way communication system where the client sends a message to the server, the server processes the message by decrementing the first character of the string, displays it, and then responds based on user input. The client receives this response and again decrements the first character before displaying it. This interaction continues until either the server or the client sends the message "Stop", at which point the communication terminates and the connection closes. The programs demonstrate the use of key networking concepts such as socket creation, data input/output streams, string handling, and interactive data exchange, making them a practical example of basic bidirectional socket communication in Java.

Algorithm:

For Server Program –

Step 1: Start the server.

Step 2: Create a server socket on port 1234.

Step 3: Wait for a client to connect and accept the connection.

Step 4: Initialize input (DataInputStream) and output (DataOutputStream) streams to communicate with the client.

Step 5: Initialize a BufferedReader to read input from the server user (keyboard).

Step 6: Declare two strings: str and str2 (used for receiving and sending messages).

Step 7: Repeat the following steps in a loop until the received string str equals "Stop":

- Read the string str from the client.
- If str is "Stop":
    - Set str2 = "Stop".
- Else:
    - Extract the first character of str.
    - Decrement the character by 1 (e.g., 'B' becomes 'A').
    - Print the decremented character.
    - Read a reply string (str2) from the server user via keyboard input.
- Send str2 to the client.
- Flush the output stream.

Step 8: Close the input stream, client socket, and server socket.

Step 9: End.

For Client Program –

Step 1: Start the client.

Step 2: Create a socket and connect to the server at localhost on port 1234.

Step 3: Initialize:

- DataInputStream to receive data from the server.
- DataOutputStream to send data to the server.
- BufferedReader to read user input from the keyboard.

Step 4: Declare two string variables: str and str2.

Step 5: Repeat the following steps in a loop until the user input str equals "Stop":

- Read a line from the user and assign it to str.
- Send str to the server using writeUTF() and flush the stream.
- Receive a response string str2 from the server using readUTF().
- If the received string str2 is "Stop":
  - Break the loop.
- Else:
  - Extract the first character of str2.
  - Decrement the character by 1 (e.g., 'D' becomes 'C').
  - Print the decremented character.

Step 6: After exiting the loop:

- Flush and close the output stream.
- Close the socket connection.

Step 7: End.

Implementation:

For Server Program –

```
import java.net.*;

import java.io.*;

public class Serverabcdrev{

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

ServerSocket ss = new ServerSocket(1234);

Socket s = ss.accept();

DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
String str = "", str2 = "";
while(!str.equals("Stop")){
str = din.readUTF();
if(str.equals("Stop"))
{
str2 = str;
}
else
{
char ch = str.charAt(0);
ch-=1;
System.out.println(ch);
str2 = br.readLine();
}
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}
}
```

For Client Program –

```
import java.net.*;
import java.io.*;
public class Clientabcdrev{
public static void main(String arg[]) throws Exception{
//InetAddress ia = InetAddress.getLocalHost();
Socket s = new Socket("localhost",1234);
```

SELECTIVE REPEAT ARQ

Problem Description: Develop a simulation of the Selective Repeat Automatic Repeat Request (ARQ) protocol using Java socket programming. The system is divided into two parts: a server (SRs) and a client (SRc). The client takes input from the user in the form of a specified number of data frames and transmits them to the server through a socket connection. Upon receiving the frames, the server simulates an error detection mechanism by checking for any frame with a value of -1, which represents a lost or corrupted frame. If such a frame is found, the server requests a retransmission of that specific frame by sending its index back to the client. The client then retransmits the requested frame, which the server receives and replaces in the original sequence. This approach closely models the behavior of the Selective Repeat ARQ, where only the erroneous or lost frame is retransmitted instead of the entire sequence, thus improving efficiency. The connection is closed after the frame exchange and retransmission process is complete. This simulation helps in understanding how selective retransmission works in reliable data communication protocols.

Algorithm:

For Server Program –

Step 1: Start the server.

Step 2: Create a ServerSocket on port 8011.

Step 3: Display a message: "Waiting for connection".

Step 4: Accept the connection request from a client using accept().

Step 5: Create DataInputStream and DataOutputStream objects to read from and write to the client.

Step 6: Read the total number of frames (numOfFrames) sent by the client.

Step 7: Initialize an integer array frames[] of size numOfFrames.

Step 8: Loop from i = 0 to i < numOfFrames:

- Read each frame value using readInt().
- Store it in the frames array.
- Display the frame value.

Step 9: Call the method requestRetransmission():

- Search for any frame in the array with the value -1.
- If found, return its index.
- If no such frame exists, return -1.

Step 10: Send the index of the frame to be retransmitted back to the client using writeInt().

Step 11: If the returned index is not -1:

- Read the retransmitted frame from the client.
- Replace the corrupted frame in the array with the new value.

- Display a message showing the corrected frame and its position.

Step 12: Display "Closing connection".

Step 13: Close all resources: DataInputStream, DataOutputStream, and ServerSocket.

Step 14: End.

For Client Program –

Step 1: Start the client.

Step 2: Create a Scanner object to read input from the user.

Step 3: Resolve the IP address of the server using InetAddress.getByName("localhost").

Step 4: Establish a connection to the server on port 8011 using a Socket.

Step 5: Create DataOutputStream and DataInputStream objects to send and receive data through the socket.

Step 6: Prompt the user to enter the total number of frames to be sent.

Step 7: Read the number of frames (numOfFrames) from the user.

Step 8: Send numOfFrames to the server using writeInt() and flush the stream.

Step 9: Prompt the user to enter the individual frame values.

Step 10: Loop from i = 0 to i < numOfFrames:

- Read each frame value from the user.
- Send the frame value to the server using writeInt() and flush the stream.

Step 11: Wait for the server's response indicating if any frame needs retransmission:

- Read the index of the requested frame using readInt().

Step 12: If the server requests a retransmission (i.e., index is not -1):

- Display the requested frame number to the user.
- Read the new frame value from the user.
- Send the retransmitted frame to the server using writeInt() and flush the stream.

Step 13: Display "Closing connection".

Step 14: Close the socket connection and the scanner.

Step 15: End.

Implementation:

For Server Program –

```
import java.io.*;

import java.net.*;
```

```
public class SRs {
static ServerSocket serverSocket;
static DataInputStream dis;
static DataOutputStream dos;
static int[] frames;
public static void main(String[] args) {
try {
serverSocket = new ServerSocket(8011);
System.out.println("Waiting for connection");
Socket client = serverSocket.accept();
dis = new DataInputStream(client.getInputStream());
dos = new DataOutputStream(client.getOutputStream());
int numOfFrames = dis.readInt();
frames = new int[numOfFrames];
System.out.println("Received " + numOfFrames + " frames from client:");
for (int i = 0; i < numOfFrames; i++) {
frames[i] = dis.readInt();
System.out.println(frames[i]);
}
int request = requestRetransmission();
dos.writeInt(request);
dos.flush();
if (request != -1) {
int retransmitFrame = dis.readInt();
frames[request] = retransmitFrame;
System.out.println("Received retransmitted frame " + retransmitFrame + " for frame " + (request +
1));
}
System.out.println("Closing connection");
} catch (IOException e) {
System.out.println(e);
} finally {
try {
dis.close();
dos.close();
serverSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
static int requestRetransmission() {
for (int i = 0; i < frames.length; i++) {
if (frames[i] == -1) {
return i;
}
}
return -1;
}
}
```

For Client Program –

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
public class SRc {
static Socket connection;
public static void main(String a[]) {
try {
Scanner scanner = new Scanner(System.in);
InetAddress addr = InetAddress.getByName("localhost");
System.out.println(addr);
connection = new Socket(addr, 8011);
DataOutputStream out = new DataOutputStream(connection.getOutputStream());
DataInputStream in = new DataInputStream(connection.getInputStream());
System.out.print("Enter the number of frames to send: ");
int numOfFrames = scanner.nextInt();
out.writeInt(numOfFrames);
out.flush();
System.out.println("Enter the frames:");
for (int i = 0; i < numOfFrames; i++) {
int frame = scanner.nextInt();
out.writeInt(frame);
out.flush();
}
int request = in.readInt();
if (request != -1) {
System.out.println("Server requests retransmission for frame " + (request + 1));
int retransmitFrame = scanner.nextInt();
out.writeInt(retransmitFrame);
out.flush();
}
System.out.println("Closing connection");
connection.close();
scanner.close();
} catch (IOException e) {
System.out.println(e);
}
}
}
```

DOMAIN NAME SYSTEM

Problem Description: Develop basic Domain Name System (DNS) functionality by providing a console-based application that allows users to perform both DNS and reverse DNS lookups. By leveraging Java's networking classes, particularly InetAddress, the program enables users to enter a hostname and retrieve its corresponding IP address, or input an IP address to obtain the associated hostname. The application operates through a menu-driven interface, prompting users for their choice and input, and continues running until the user opts to exit. It also includes appropriate exception handling to manage invalid inputs or lookup failures. This project serves as an educational tool to help understand how DNS resolution works in computer networks.

Algorithm:

Step 1: Start the program.

Step 2: Create a BufferedReader object to take user input from the console.

Step 3: Repeat the following steps until the user chooses to exit:

- Display a menu with options:
    1. Display
    2. Reverse DNS Lookup
    3. Exit
- Prompt the user to enter their choice and read the input.
- If the user selects option 1 (DNS Lookup):
    - Prompt the user to enter a hostname.
    - Use InetAddress.getByName(hostname) to resolve the hostname to an IP address.
    - Display the hostname and its corresponding IP address.
- Else if the user selects option 2 (Reverse DNS Lookup):
    - Prompt the user to enter an IP address.
    - Use InetAddress.getByName(ip) to resolve the IP to a hostname.
    - Display the IP address and its associated hostname.
- Handle any IOException that may occur during input or resolution.

Step 4: End the loop when the user selects option 3 (Exit).

Step 5: Terminate the program.

Implementation:

```
import java.net.*;

import java.io.*;
import java.util.*;

public class DNS

{

public static void main(String[] args)

{

int n;

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

do

{

System.out.println("\n Menu: \n 1. DNS 2. Reverse DNS 3. Exit \n");

System.out.println("\n Enter your choice");

n = Integer.parseInt(System.console().readLine()); 2

if(n==1)

{

try

{

System.out.println("\n Enter Host Name ");

String hname=in.readLine();

InetAddress address;

address = InetAddress.getByName(hname);

System.out.println("Host Name: " + address.getHostName());

System.out.println("IP: " + address.getHostAddress());

}

catch(IOException ioe)

{

ioe.printStackTrace();

}

}

if(n==2)

{

try
```

REVERSE, CONCATINATION, UPPER, LOWER, SENTENCE

Problem Description: Develop a client-server application in Java where the client sends string-based requests to the server, and the server processes these requests and responds accordingly. The server listens on a specific port and handles various string operations such as reversing a string, concatenating two strings, converting a string to uppercase, converting a string to lowercase, and formatting a string into sentence case. Based on the client's selected option, corresponding string input is sent to the server, which processes the data and returns the result to the client for display. This application demonstrates the use of TCP sockets, data streams, and string manipulation in a two-way communication system, reinforcing the fundamentals of distributed systems and real-time client-server interaction.

Algorithm:

For Server Program –

Step 1: Start the program.

Step 2: Create a ServerSocket on port 8888 to listen for client connections.

Step 3: Accept the client connection using accept() method and create input (DataInputStream) and output (DataOutputStream) streams for communication.

Step 4: Display a message indicating successful connection.

Step 5: Repeat the following steps until the client sends the exit option (op = 6):

- Read the option (op) from the client using readUTF() and convert it to an integer.
- Initialize two empty strings str1 and str2.
- Use a switch statement to handle different string operations based on the value of op:
    - Case 1 (Reverse):
        - Read a string from the client.
        - Reverse the string by prepending each character to str2.
        - Send the reversed string back to the client.
    - Case 2 (Concatenation):
        - Read two strings from the client.
        - Concatenate them with a space in between.
        - Send the concatenated string back to the client.
    - Case 3 (Uppercase):
        - Read a string from the client.
        - Convert the string to uppercase.
        - Send the uppercase string back to the client.

    - Case 4 (Lowercase):
        - Read a string from the client.
        - Convert the string to lowercase character by character.
        - Send the lowercase string back to the client.
    - Case 5 (Sentence Case):
        - Read a string from the client.
        - Convert the first character to uppercase and the rest to lowercase.
        - Send the sentence-case string back to the client.
    - Case 6 (Exit):
        - Exit the loop.
    - Default:
        - Print a message indicating an invalid option.

Step 6: After exiting the loop, close the input and output streams, the client socket, and the server socket.

Step 7: End the program.

For Client Program –

Step 1: Start the program.

Step 2: Establish a connection to the server running on localhost and port 8888 using a Socket.

Step 3: Create input and output streams:

- DataInputStream to receive data from the server.
- DataOutputStream to send data to the server.
- BufferedReader to read input from the user via keyboard.

Step 4: Repeat the following steps until the user selects the Exit option (op = 6):

- Display the menu with the following options:
    1. Reverse
    2. Concatenate
    3. Uppercase
    4. Lowercase
    5. Sentence case
    6. Exit
- Prompt the user to enter their choice and read it.

- Send the selected option to the server using writeUTF().
- Initialize two strings: str1 and str2.
- Use a switch statement to handle each option:
    - Case 1 (Reverse):
        - Prompt the user to enter a string.
        - Send the string to the server.
        - Receive the reversed string from the server and display it.
    - Case 2 (Concatenate):
        - Prompt the user to enter two strings.
        - Send both strings to the server.
        - Receive the concatenated string from the server and display it.
    - Case 3 (Uppercase):
        - Prompt the user to enter a string.
        - Send the string to the server.
        - Receive the uppercase version from the server and display it.
    - Case 4 (Lowercase):
        - Prompt the user to enter a string.
        - Send the string to the server.
        - Receive the lowercase version from the server and display it.
    - Case 5 (Sentence case):
        - Prompt the user to enter a string.
        - Send the string to the server.
        - Receive the sentence-cased string from the server and display it.
    - Case 6 (Exit):
        - Exit the loop.
    - Default:
        - Display an error message for invalid input.

Step 5: After exiting the loop, close the output stream and the socket connection.

Step 6: End the program.

Implementation:

For Server Program –

```
import java.net.ServerSocket;

import java.net.Socket;

import java.io.*;


class Serverstr {

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

ServerSocket ss = new ServerSocket(8888);

Socket s = ss.accept();

DataInputStream din = new DataInputStream(s.getInputStream());

DataOutputStream dout = new DataOutputStream(s.getOutputStream());

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

int op;

do {

String sop = din.readUTF();

op = Integer.parseInt(sop);

System.out.println("Option given by the client: " + op);

String str1 = "", str2 = "";

switch (op) {

case 1:

str1 = din.readUTF();

System.out.println("String given by the client: " + str1);

for (int i = 0; i < str1.length(); i++) {

str2 = str1.charAt(i) + str2;

}

dout.writeUTF(str2);

dout.flush();

break;

case 2:

str1 = din.readUTF();

System.out.println("String given by the client: " + str1);
```

```
str2 = din.readUTF();
System.out.println("String given by the client: " + str2);
str2 = str1 + " " + str2;
dout.writeUTF(str2);
dout.flush();
break;
case 3:
str1 = din.readUTF();
System.out.println("String given by the client: " + str1);
str2 = str1.toUpperCase();
dout.writeUTF(str2);
dout.flush();
break;
case 4:
str1 = din.readUTF();
System.out.println("String given by the client: " + str1);
str1 = str1.toLowerCase();
char[] arr = str1.toCharArray();
for (int i = 0; i < arr.length; i++) {
arr[i] = Character.toLowerCase(arr[i]);
}
str2 = new String(arr);
dout.writeUTF(str2);
dout.flush();
break;
case 5:
str1 = din.readUTF();
System.out.println("String given by the client: " + str1);
if (!str1.isEmpty()) {
str2 = Character.toUpperCase(str1.charAt(0)) + str1.substring(1).toLowerCase();
}
```

```
dout.writeUTF(str2);
dout.flush();
break;
case 6:
break;
default:
System.out.println("WRONG INPUT");
break;
}
} while (op != 6);
din.close();
s.close();
ss.close();
}
}
```

For Client Program –

```
import java.net.Socket;
import java.io.*;
class Clientstr {
public static void main(String args[]) throws Exception {
Socket s = new Socket("localhost", 8888);
DataInputStream din = new DataInputStream(s.getInputStream());
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
int op;
do {
System.out.println("1-Reverse");
System.out.println("2-Concate");
System.out.println("3-Uppercase");
System.out.println("4-Lowercase");
System.out.println("5-Sentence");
System.out.println("6-Exit");

System.out.println("Enter your Option");

String sop = br.readLine();

op = Integer.parseInt(sop);

dout.writeUTF(sop);

String str1 = "", str2 = "";

switch (op) {

case 1:

System.out.println("Enter a string");

str1 = br.readLine();

dout.writeUTF(str1);

str2 = din.readUTF();

System.out.println("Reverse: " + str2);

dout.flush();

break;

case 2:

System.out.println("Enter a string");

str1 = br.readLine();

System.out.println("Enter a string");

str2 = br.readLine();

dout.writeUTF(str1);

dout.writeUTF(str2);

dout.flush();

str2 = din.readUTF();

System.out.println("Concate: " + str2);

break;

case 3:

System.out.println("Enter a String: ");

str1 = br.readLine();

dout.writeUTF(str1);

str2 = din.readUTF();
System.out.println("Uppercase: " + str2);
break;
case 4:
System.out.println("Enter a String: ");
str1 = br.readLine();
dout.writeUTF(str1);
str2 = din.readUTF();
System.out.println("Lowercase: " + str2);
break;
case 5:
System.out.println("Enter a String: ");
str1 = br.readLine();
dout.writeUTF(str1);
str2 = din.readUTF();
System.out.println("Sentence: " + str2);
break;
case 6:
break;
default:
System.out.println("WRONG INPUT");
break;
}
} while (op != 6);
dout.close();
s.close();
}
}
```