**Experiment no – 04**

**Aim**:

Write a client–server application using Java sockets that allows a client to write a message (as a String) to a socket. A server will read this message, count the number of characters and digits in the message, and send these two counts back to the client. The server will listen to port 6100. The client can obtain the String message that it is to pass to the server either from the command line or by using a prompt to the user. One strategy for sending the two counts back to the client is for the server to construct an object containing:

1. The message it receives from the client
2. A count of the number of characters in the message
3. A count of the number of digits in the message.

**Theory:**

The term "client/server" describes a type of distributed processing in which an application is divided into two parts, each possibly residing on separate operating systems, but working together to provide a service to the end user. As shown in Figure 1, one part of the application, the client, typically resides on a workstation and requests a service for the end-user. The other part of the application, the server, usually resides on a larger machine, such as a mainframe computer. The server program uses the resources of the mainframe computer to perform services requested by each client.

Depending on how it is designed, a server can process requests from multiple clients concurrently. Generally, there is one server for many clients.

The client is usually the part of the application that is "seen" by the end-user. Therefore, the client half of a client/server application most often resides on a workstation, where the end-user can interact with the application through the workstation operating system's graphical user interface.

Servers, on the other hand, are usually transparent to the end-user. That is, the person who sits at the workstation only perceives the client half of the application, the part that displays the information (though it was actually retrieved by a remote server). Because there is only one server for a given set of clients, servers provide an ideal way of managing shared access to system resources, such as data sets. For this reason, servers are likely to reside on larger machines such as z/OS mainframe computers.

Usually, the same person writes both the client and server parts of a client/server application.

**Code:**

**Server2.java**

package server2;

import java.net.\*;

import java.io.\*;

public class Server2 {

private static Socket socket;

public static void main(String[] args) {

try {

int port = 6102;

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server Started and listening to the port 6102");

//Server is running always. This is done using this while(true) loop

while (true) {

//Reading the message from the client

socket = serverSocket.accept();

InputStream is = socket.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String msg = br.readLine();

System.out.println("Message received from Yash Prajapati - 022 Client is " + msg);

String returnMessage = "";

try {

int d = 0;

int v = 0;

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

char k = msg.charAt(i);

if (k >= '0' && k <= '9') {

d++;

} else if (k == 'a' || k == 'e' || k == 'i' || k == 'o' || k == 'u') {

v++;

}

}

String ans = "No.of characters = " + msg.length() + "; No. of vovels = " + v + "; No.of digits= " + d;

returnMessage = String.valueOf(ans) + "\n";

} catch (Exception e) {

}

//Sending the response back to the client.

OutputStream os = socket.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

bw.write(returnMessage);

System.out.println("Message sent to the client is " + returnMessage);

bw.flush();

}

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

socket.close();

} catch (Exception e) {

}

}

}

}

**Client2.java**

package client2;

import java.net.\*;

import java.io.\*;

public class Client2

{

private static Socket socket;

public static void main(String args[])

{

try

{

String host = "localhost";

int port = 6102;

InetAddress address = InetAddress.getByName(host);

socket = new Socket(address, port);

//Send the message to the server

OutputStream os = socket.getOutputStream();

OutputStreamWriter osw = new OutputStreamWriter(os);

BufferedWriter bw = new BufferedWriter(osw);

String msg = "Computer Science 2022";

String sendMessage = msg + "\n";

bw.write(sendMessage);

bw.flush();

System.out.println("Message sent to the server : "+sendMessage);

//Get the return message from the server

InputStream is = socket.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String message = br.readLine();

System.out.println("Message received from Yash Prajapati - 022 the server : " +message);

}

catch (Exception exception)

{

exception.printStackTrace();

}

finally

{

//Closing the socket

try

{

socket.close();

}

catch(Exception e)

{

e.printStackTrace();

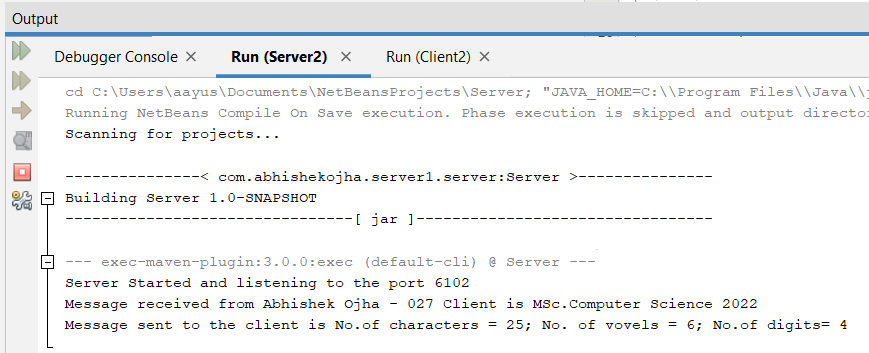
}

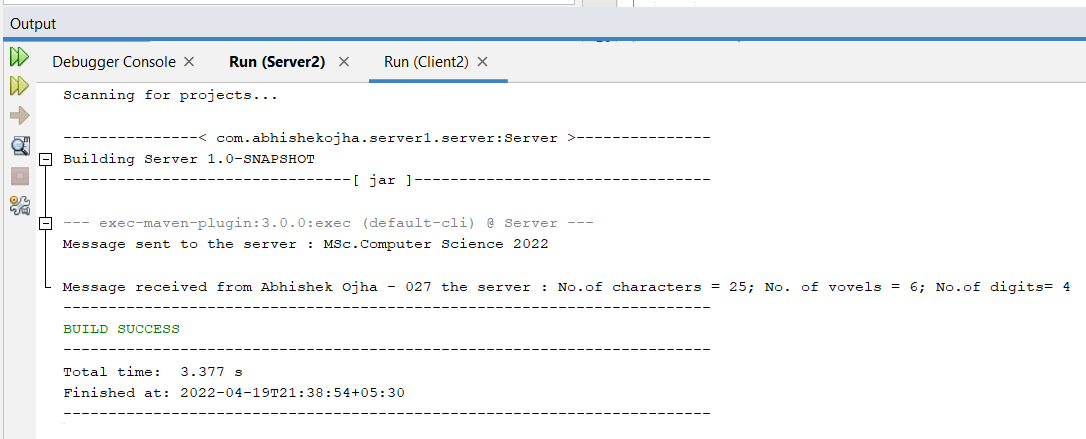
}

}

}

**Output:**

****

****

**Conclusion:**

Successfully demonstrated a client server program to send and receive message.

.