With the support of the Erasmus+ Programme of the European Union

EMJMD GENIAL

GREEN NETWORKING & CLOUD COMPUTING

LULEÅ TEKNISKA UNIVERSITET

Lab 2: Fundamentals of network programming with JAVA & JAVA network security

MODULE: Network programming and distributed applications

Authors: Mirjalol Aminov, Nadir Arfi, Chandan Singh

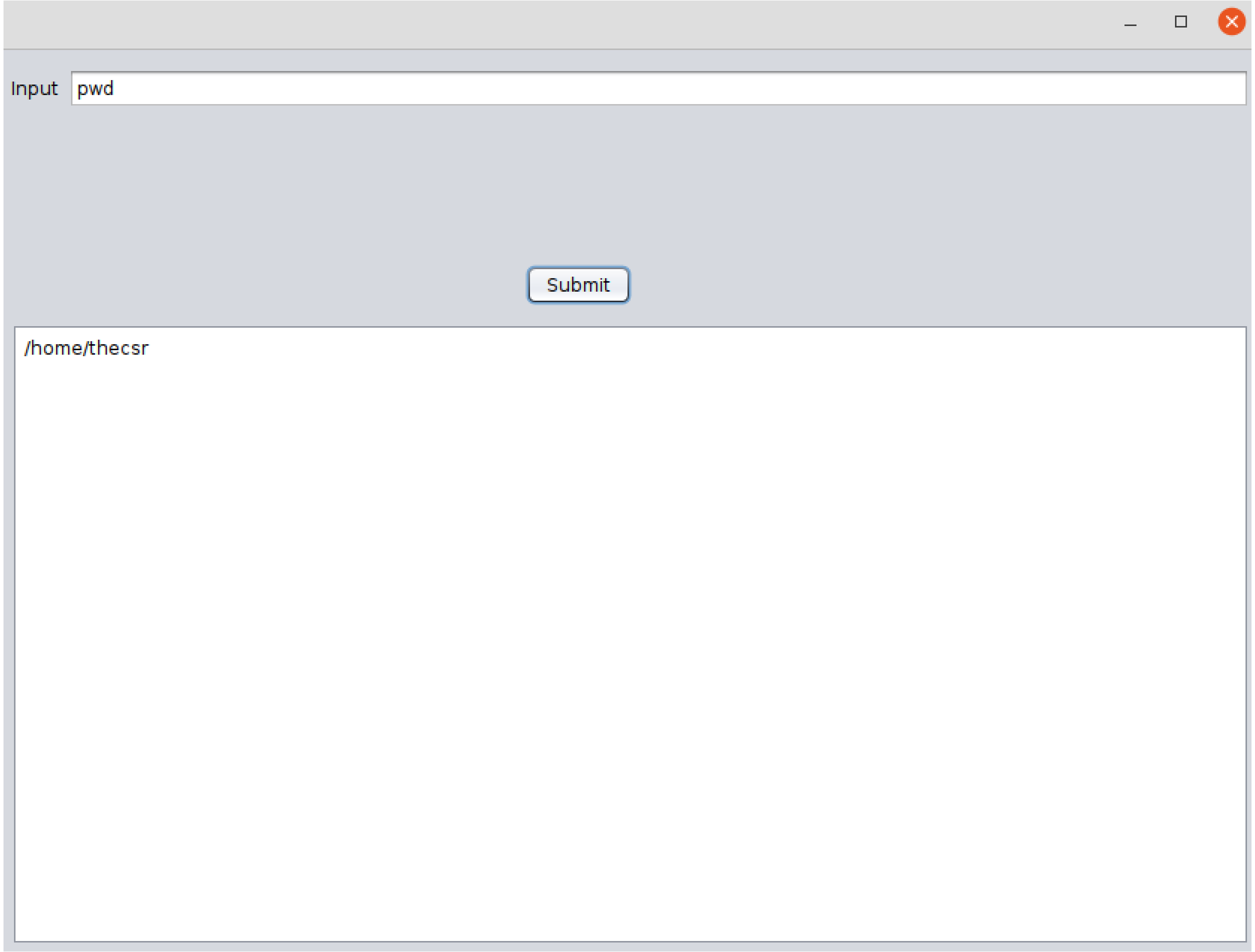
Supervisor: Pr. Evgeniy Osipov

University: Luleå University of Technology, Sweden

Program: EMJMD GENIAL

Part One: Java GUI

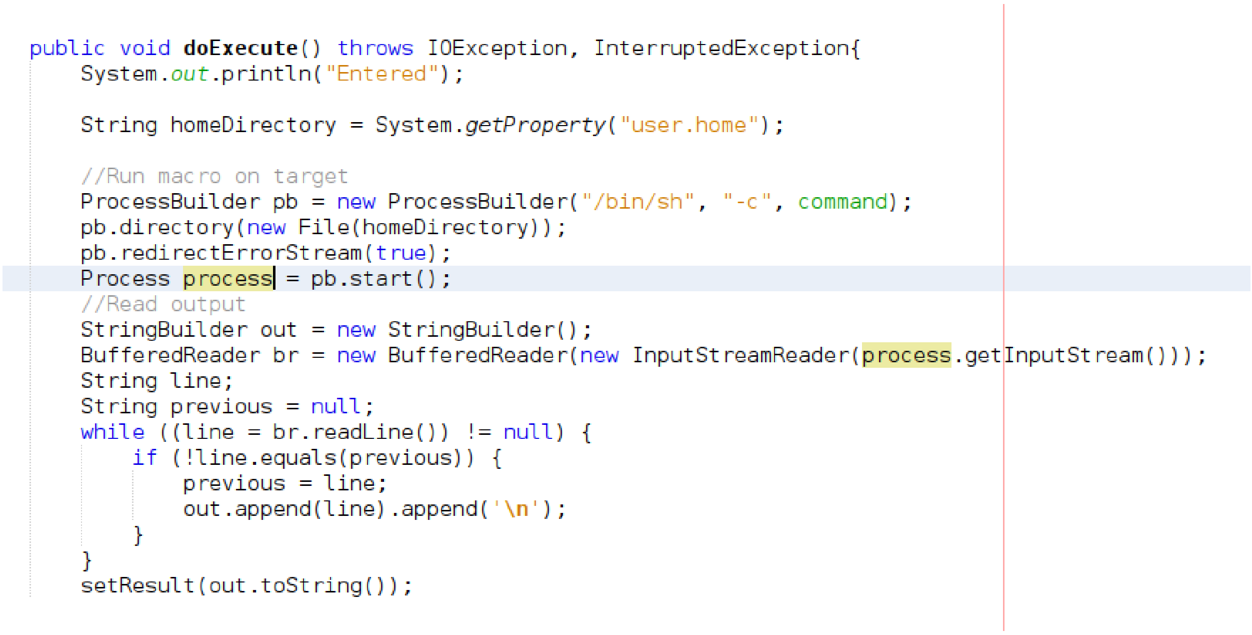
Java's Swing is a simple GUI toolkit with a large selection of widgets for creating window-based applications that are efficient. It belongs to the JFC (Java Foundation Classes). It is totally developed in Java and built around the AWT API. In contrast to AWT, it is platform independent and contains lightweight components.In Part 1 of the lab, we will be using Swing to build the GUI for our application. The Main objective of this part of the lab is to build a User interface using java to run the unix/windows/mac commands. The IDE we select to run swing is Netbeans because of its feature to drag and drop the swing gui components.
On the User interface front we need to create a text field that takes in the command, a button to hande the action and then another text area that will display the result after the commands are executed. The Final GUI is shown in the below figure.



Input pwd

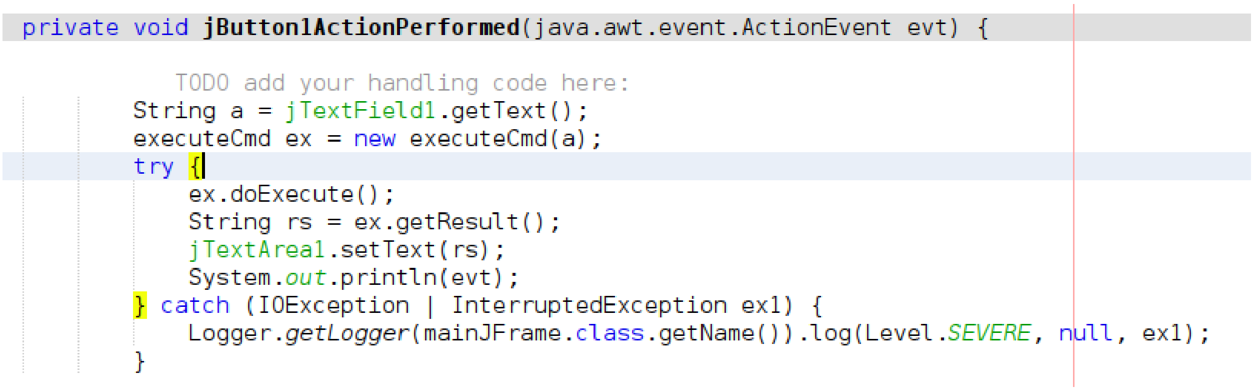
```
/home/thecsr
```

To run the unix commands we use the Procesbuilder, which can be used to create operating system processes.



```
public void doExecute() throws IOException, InterruptedException{
    System.out.println("Entered");

    String homeDirectory = System.getProperty("user.home");

    //Run macro on target
    ProcessBuilder pb = new ProcessBuilder("/bin/sh", "-c", command);
    pb.directory(new File(homeDirectory));
    pb.redirectErrorStream(true);
    Process process = pb.start();
    //Read output
    StringBuilder out = new StringBuilder();
    BufferedReader br = new BufferedReader(new InputStreamReader(process.getInputStream()));
    String line;
    String previous = null;
    while ((line = br.readLine()) != null) {
        if (!line.equals(previous)) {
            previous = line;
            out.append(line).append('\n');
        }
    }
    setResult(out.toString());
```

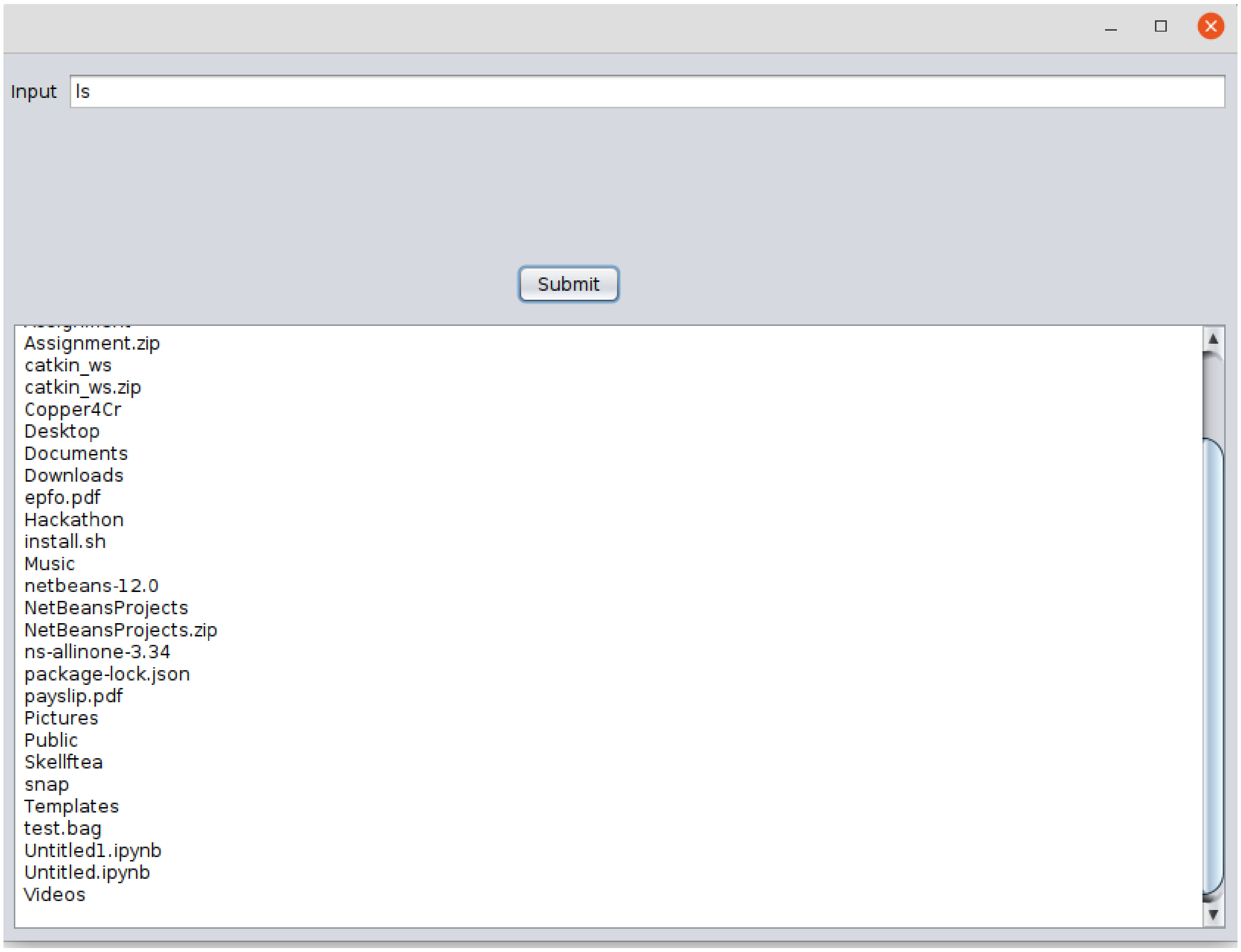
To use the process builder class,we just instantiate the class and pass the arguments “bin/sh”, “-c” and the command that need to be executed. After the argument as are passed, using the object of the class, we pass the directory in which this command should be run to the processbuilder.After that we start the process. We use StringBuilder class to finally print the final result. StringBuilder is used because it allows to create a mutuable sequence of characters.We further use the “getter” and “setter” in our code to get the result and set the command.



```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

           TODO add your handling code here:
        String a = jTextField1.getText();
        executeCmd ex = new executeCmd(a);
        try {
            ex.doExecute();
            String rs = ex.getResult();
            jTextArea1.setText(rs);
            System.out.println(evt);
        } catch (IOException | InterruptedException ex1) {
            Logger.getLogger(mainJFrame.class.getName()).log(Level.SEVERE, null, ex1);
        }
```

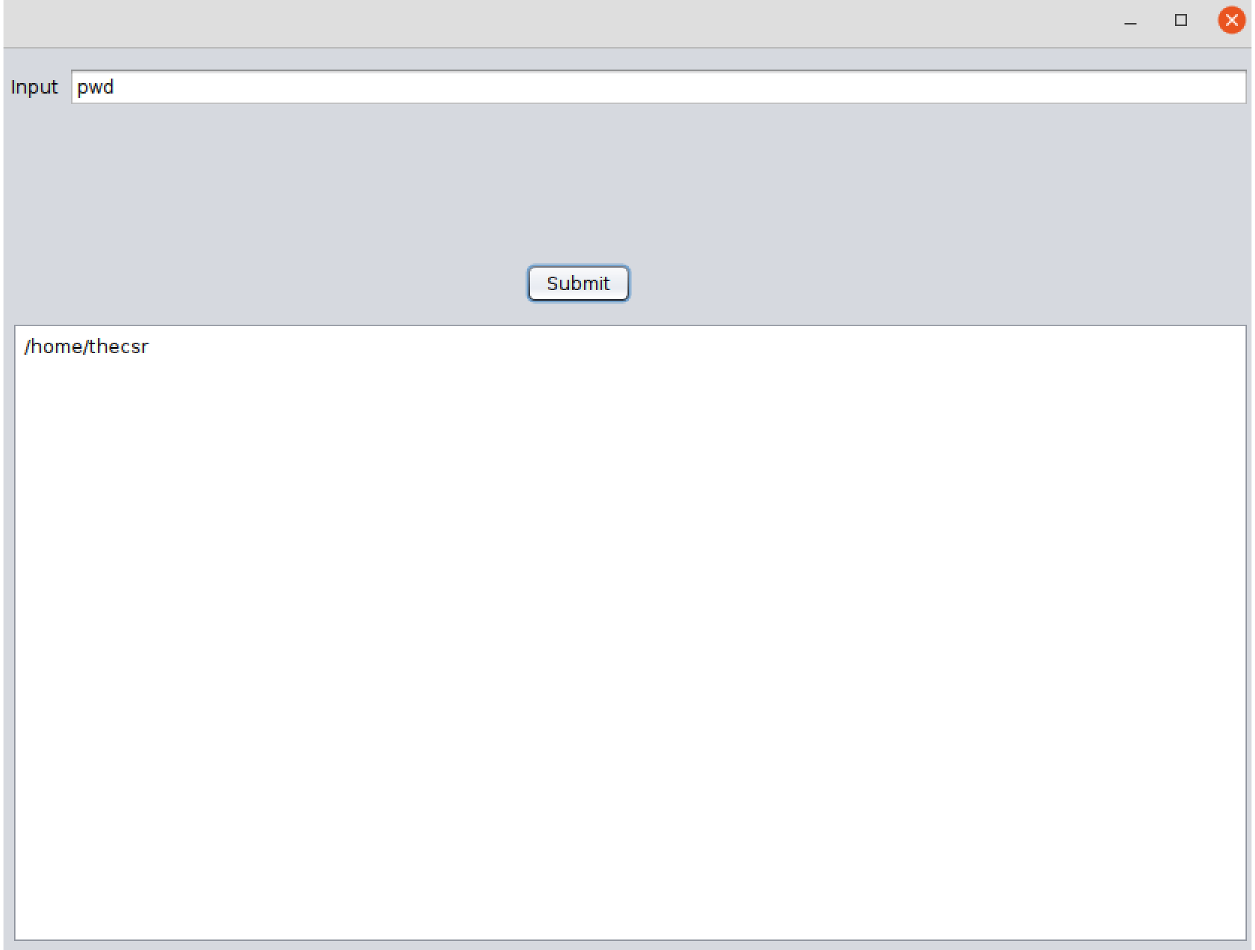
Now we call instantiate the doExecute class from the class that is generated for the gui. We call the class from the Button Action Handler method(*JButton1ActionPerformed*). In this method , what we do is take the text that is in the field1(the command field) and then pass it to the doExecute class and after the processing is done we set the result sent from the do execute class in the JtextArea1 field.

Below are the images attached, which shows different command and its result in the GUI.



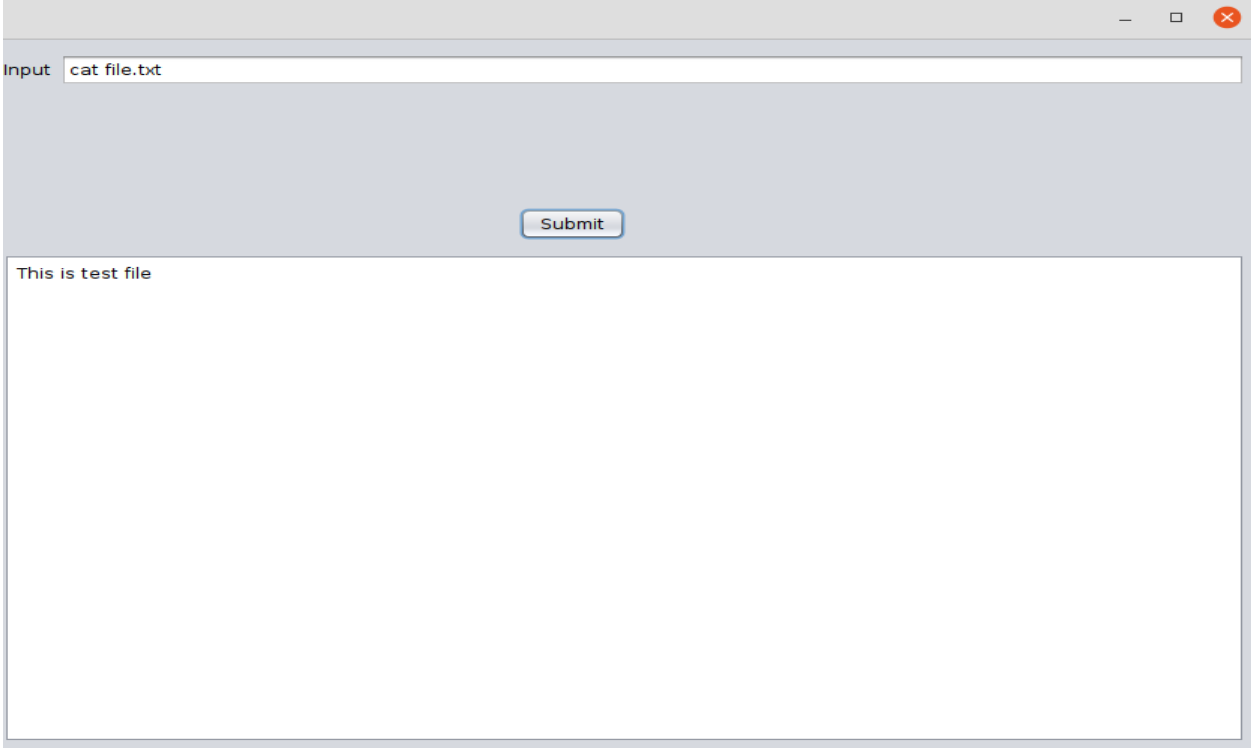
Input ls

```
Assignment.zip
catkin_ws
catkin_ws.zip
Copper4Cr
Desktop
Documents
Downloads
epfo.pdf
Hackathon
install.sh
Music
netbeans-12.0
NetBeansProjects
NetBeansProjects.zip
ns-allinone-3.34
package-lock.json
payslip.pdf
Pictures
Public
Skellftea
snap
Templates
test.bag
Untitled1.ipynb
Untitled.ipynb
Videos
```



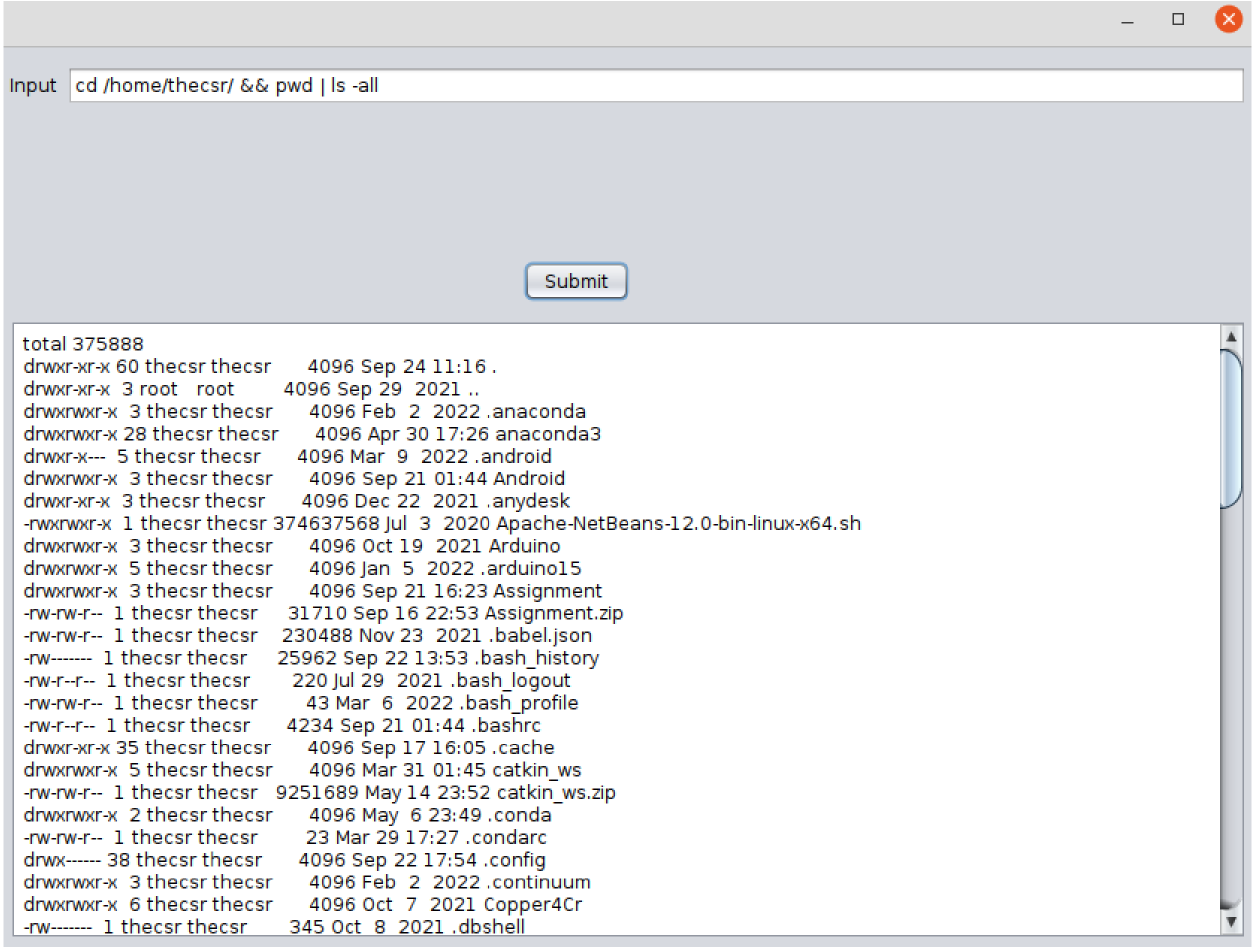
Input pwd

```
/home/thecsr
```



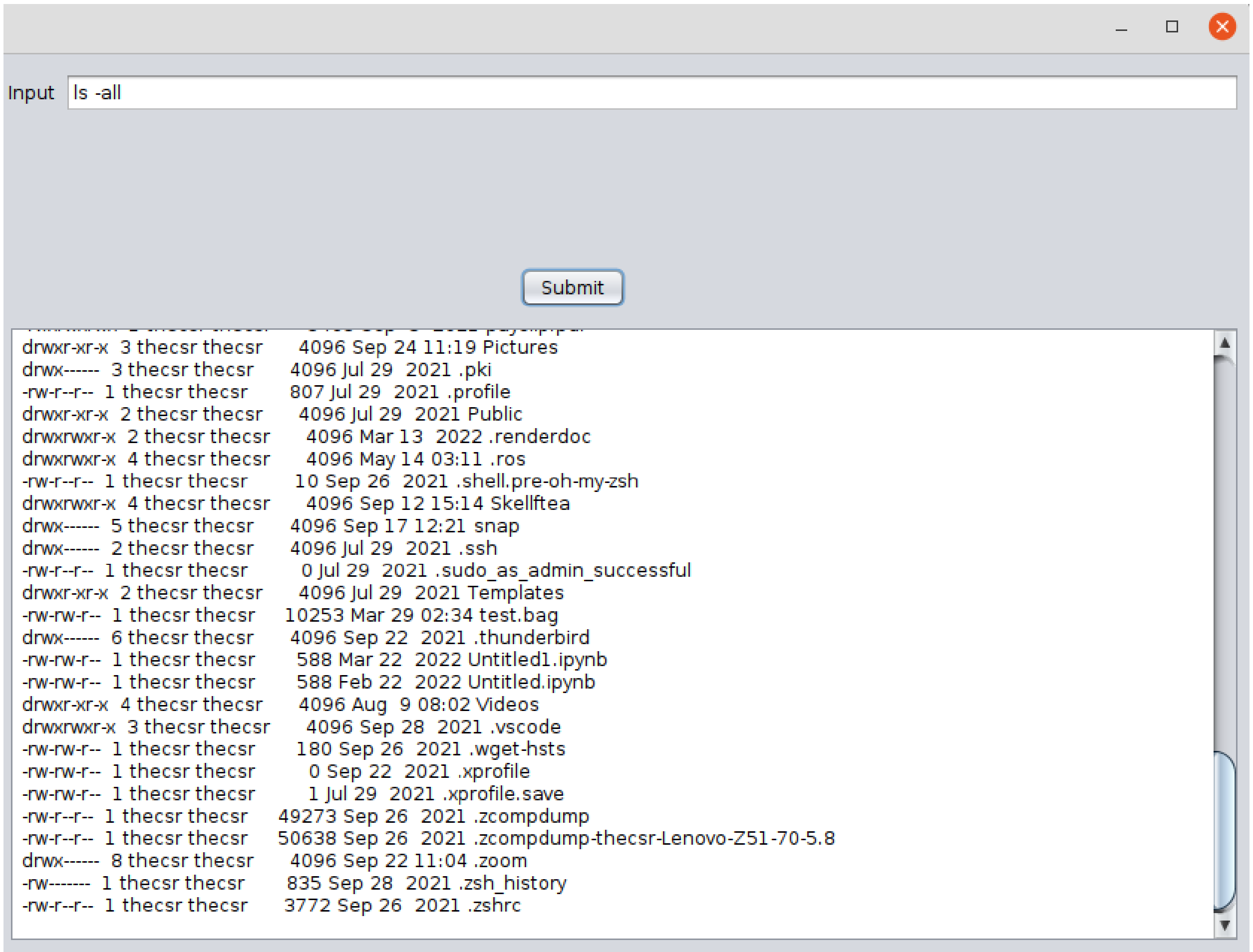
Input cat file.txt

```
This is test file
```



```
cd /home/thecsr/ && pwd | ls -all
```

```
total 375888
drwxr-xr-x 60 thecsr thecsr      4096 Sep 24 11:16 .
drwxr-xr-x  3 root   root        4096 Sep 29  2021 ..
drwxrwxr-x  3 thecsr thecsr      4096 Feb  2  2022 .anaconda
drwxrwxr-x 28 thecsr thecsr      4096 Apr 30 17:26 anaconda3
drwxr-x---  5 thecsr thecsr      4096 Mar  9  2022 .android
drwxrwxr-x  3 thecsr thecsr      4096 Sep 21 01:44 Android
drwxr-xr-x  3 thecsr thecsr      4096 Dec 22  2021 .anydesk
-rwxrwxr-x  1 thecsr thecsr 374637568 Jul  3  2020 Apache-NetBeans-12.0-bin-linux-x64.sh
drwxrwxr-x  3 thecsr thecsr      4096 Oct 19  2021 Arduino
drwxrwxr-x  5 thecsr thecsr      4096 Jan  5  2022 .arduino15
drwxrwxr-x  3 thecsr thecsr      4096 Sep 21 16:23 Assignment
-rw-rw-r--  1 thecsr thecsr     31710 Sep 16 22:53 Assignment.zip
-rw-rw-r--  1 thecsr thecsr    230488 Nov 23  2021 .babel.json
-rw-------  1 thecsr thecsr     25962 Sep 22 13:53 .bash_history
-rw-r--r--  1 thecsr thecsr       220 Jul 29  2021 .bash_logout
-rw-rw-r--  1 thecsr thecsr        43 Mar  6  2022 .bash_profile
-rw-r--r--  1 thecsr thecsr      4234 Sep 21 01:44 .bashrc
drwxr-xr-x 35 thecsr thecsr      4096 Sep 17 16:05 .cache
drwxrwxr-x  5 thecsr thecsr      4096 Mar 31 01:45 catkin_ws
-rw-rw-r--  1 thecsr thecsr   9251689 May 14 23:52 catkin_ws.zip
drwxrwxr-x  2 thecsr thecsr      4096 May  6 23:49 .conda
-rw-rw-r--  1 thecsr thecsr        23 Mar 29 17:27 .condarc
drwx------ 38 thecsr thecsr      4096 Sep 22 17:54 .config
drwxrwxr-x  3 thecsr thecsr      4096 Feb  2  2022 .continuum
drwxrwxr-x  6 thecsr thecsr      4096 Oct  7  2021 Copper4Cr
-rw-------  1 thecsr thecsr       345 Oct  8  2021 .dbshell
```



Input ls -all

```
drwxr-xr-x  3 thecsr thecsr     4096 Sep 24 11:19 Pictures
drwx------  3 thecsr thecsr     4096 Jul 29  2021 .pki
-rw-r--r--  1 thecsr thecsr      807 Jul 29  2021 .profile
drwxr-xr-x  2 thecsr thecsr     4096 Jul 29  2021 Public
drwxrwxr-x  2 thecsr thecsr     4096 Mar 13  2022 .renderdoc
drwxrwxr-x  4 thecsr thecsr     4096 May 14 03:11 .ros
-rw-r--r--  1 thecsr thecsr       10 Sep 26  2021 .shell.pre-oh-my-zsh
drwxrwxr-x  4 thecsr thecsr     4096 Sep 12 15:14 Skellftea
drwx------  5 thecsr thecsr     4096 Sep 17 12:21 snap
drwx------  2 thecsr thecsr     4096 Jul 29  2021 .ssh
-rw-r--r--  1 thecsr thecsr        0 Jul 29  2021 .sudo_as_admin_successful
drwxr-xr-x  2 thecsr thecsr     4096 Jul 29  2021 Templates
-rw-rw-r--  1 thecsr thecsr    10253 Mar 29 02:34 test.bag
drwx------  6 thecsr thecsr     4096 Sep 22  2021 .thunderbird
-rw-rw-r--  1 thecsr thecsr      588 Mar 22  2022 Untitled1.ipynb
-rw-rw-r--  1 thecsr thecsr      588 Feb 22  2022 Untitled.ipynb
drwxr-xr-x  4 thecsr thecsr     4096 Aug  9 08:02 Videos
drwxrwxr-x  3 thecsr thecsr     4096 Sep 28  2021 .vscode
-rw-rw-r--  1 thecsr thecsr      180 Sep 26  2021 .wget-hsts
-rw-rw-r--  1 thecsr thecsr        0 Sep 22  2021 .xprofile
-rw-rw-r--  1 thecsr thecsr        1 Jul 29  2021 .xprofile.save
-rw-r--r--  1 thecsr thecsr    49273 Sep 26  2021 .zcompdump
-rw-r--r--  1 thecsr thecsr    50638 Sep 26  2021 .zcompdump-thecsr-Lenovo-Z51-70-5.8
drwx------  8 thecsr thecsr     4096 Sep 22 11:04 .zoom
-rw-------  1 thecsr thecsr      835 Sep 28  2021 .zsh_history
-rw-r--r--  1 thecsr thecsr     3772 Sep 26  2021 .zshrc
```

Part Two: Sockets & Threads

Transmission Control Protocol (TCP)	reliable communication, error checking, retransmission
User Datagram Protocol (UDP)	faster, unreliable and connectionless

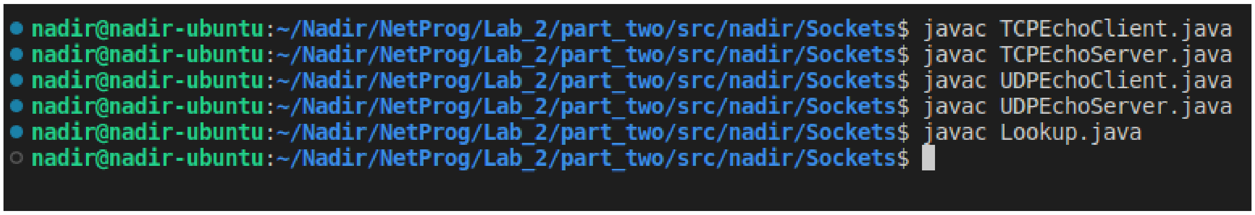
Sockets basically creates a communication channel between two nodes using either TCP or UDP. The server creates a socket object on its end of the communication by using a specific port number and it keeps listening through that port until a client attempts to connect to the server through that particular port. Once the connection is successful, both client and server can now communicate by writing and reading from the socket input and output streams. Since TCP is a two-way communication protocol, data can be sent across both streams at the same time.

Steps:

1. ServerSocket object is instantiated and denoting the port number used for connection
2. The accept() method allows server to wait until a client is connected to the port

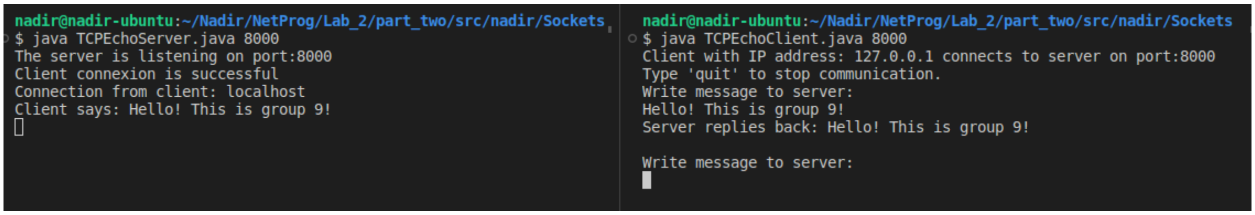
3. Client socket object is instantiated, specifying the server name and port number
4. Communication between the sockets occur using I/O streams

1. **Compile and debug the Lookup, TCPEchoServer and UDPEchoServer classes.**
   TCPEchoClient and UDPEchoClient classes are compiled as well, as shown in the figure below.



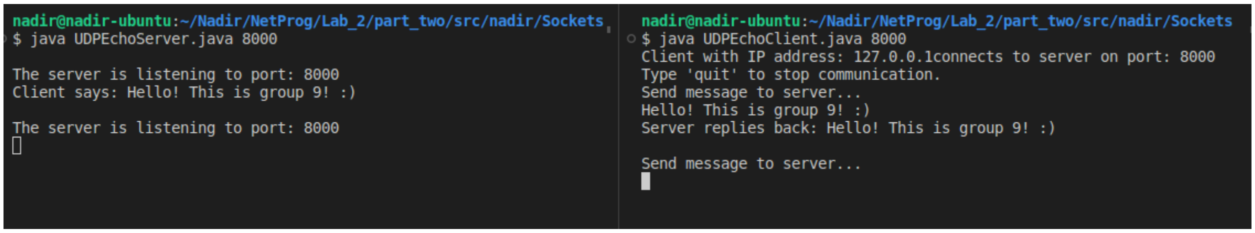
```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ javac TCPEchoClient.java
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ javac TCPEchoServer.java
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ javac UDPEchoClient.java
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ javac UDPEchoServer.java
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ javac Lookup.java
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ 
```

   Testing both TCP and UDP connections through port 8000, which is given as an argument through the terminal command.



```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets
$ java TCPEchoServer.java 8000
The server is listening on port:8000
Client connexion is successful
Connection from client: localhost
Client says: Hello! This is group 9!
```

```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets
$ java TCPEchoClient.java 8000
Client with IP address: 127.0.0.1 connects to server on port:8000
Type 'quit' to stop communication.
Write message to server:
Hello! This is group 9!
Server replies back: Hello! This is group 9!

Write message to server:
```



```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets
$ java UDPEchoServer.java 8000

The server is listening to port: 8000
Client says: Hello! This is group 9! :)

The server is listening to port: 8000
```

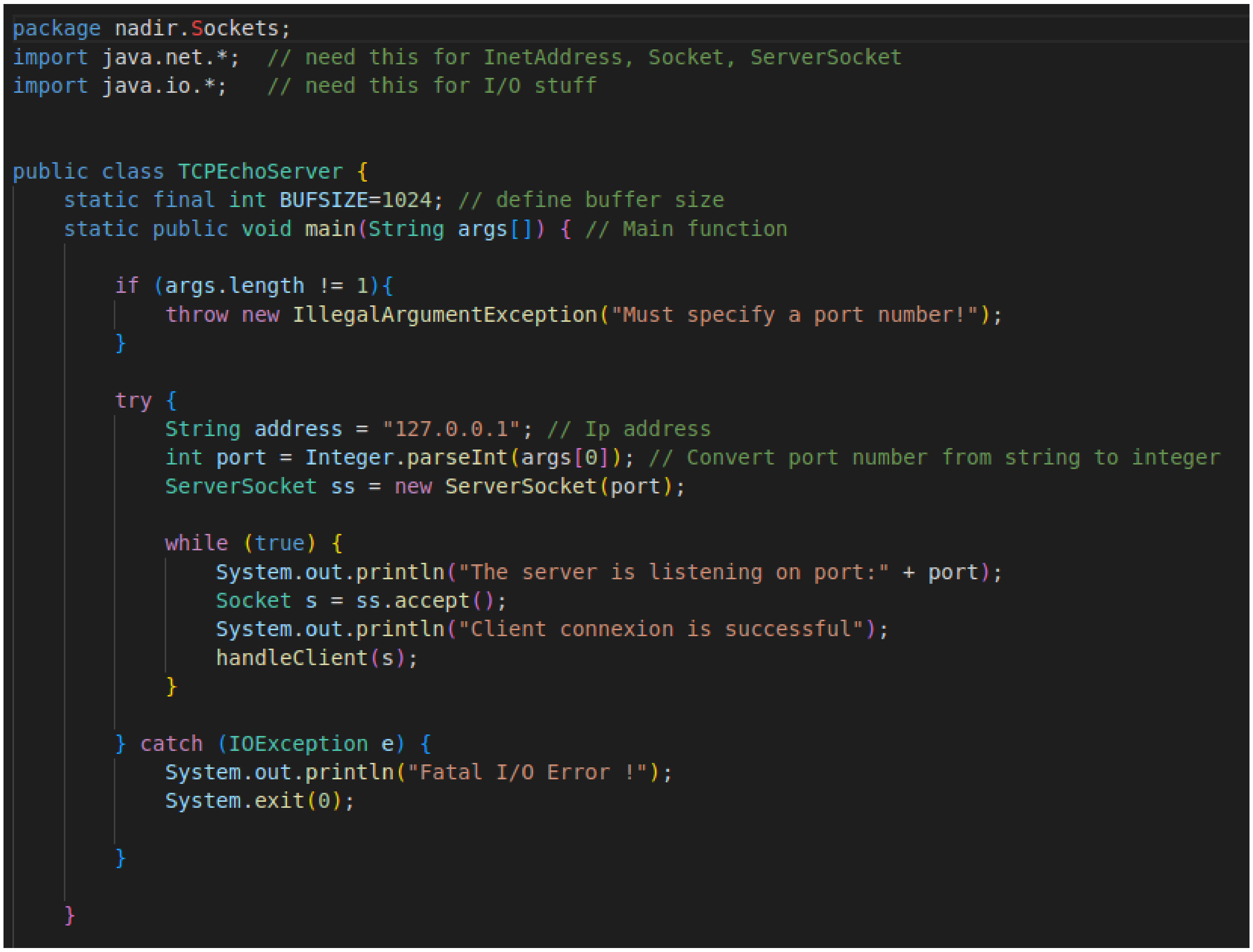
```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets
$ java UDPEchoClient.java 8000
Client with IP address: 127.0.0.1connects to server on port: 8000
Type 'quit' to stop communication.
Send message to server...
Hello! This is group 9! :)
Server replies back: Hello! This is group 9! :)

Send message to server...
```

2. **Describe the details of the implementation of each class**

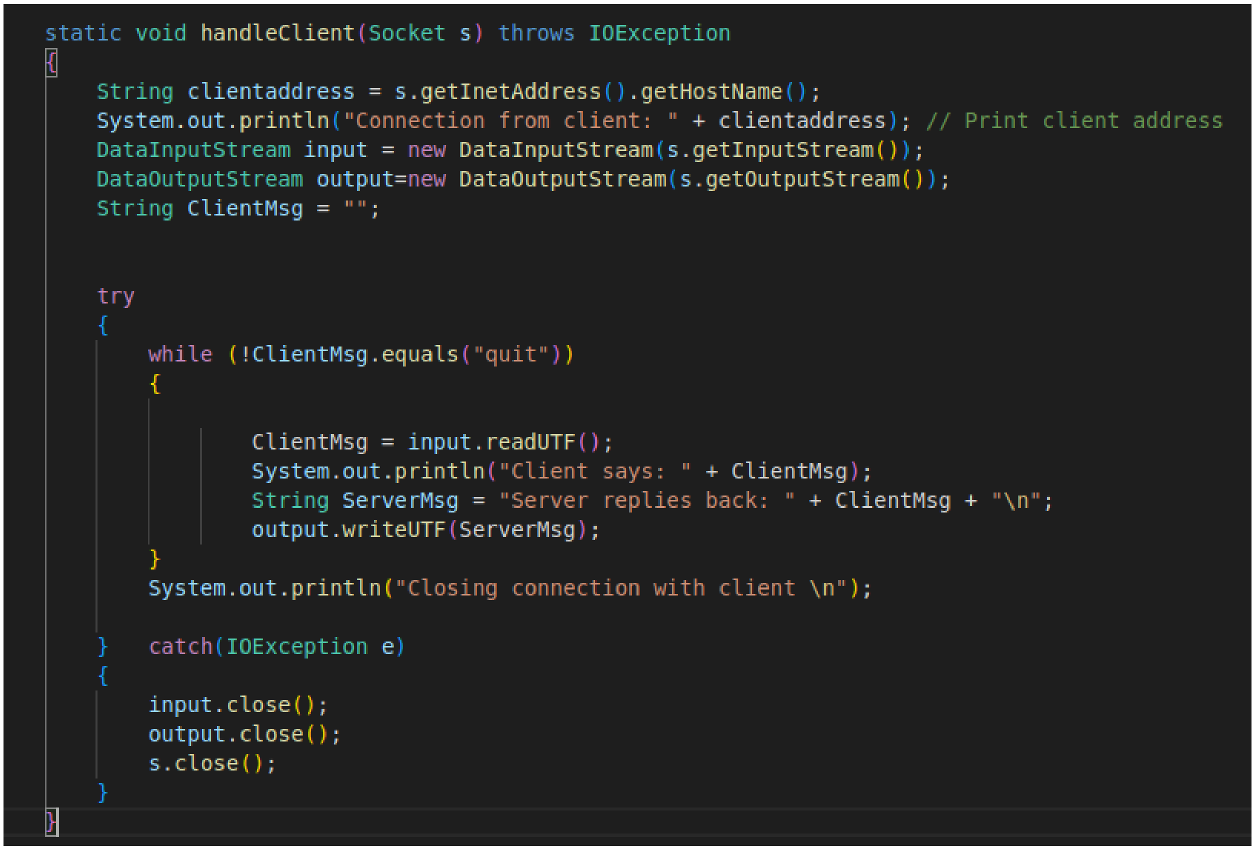
2.1. TCP Connection

2.1.1. TCPEchoServer class

- Sets a buffer size to 1024 bytes
- Check if port number is specified
- Convert port number (string to integer)
- Instantiate a server socket object
- Server invokes the accept() method, which waits until a client connects to the server on the given port
- Run a function that handles client request



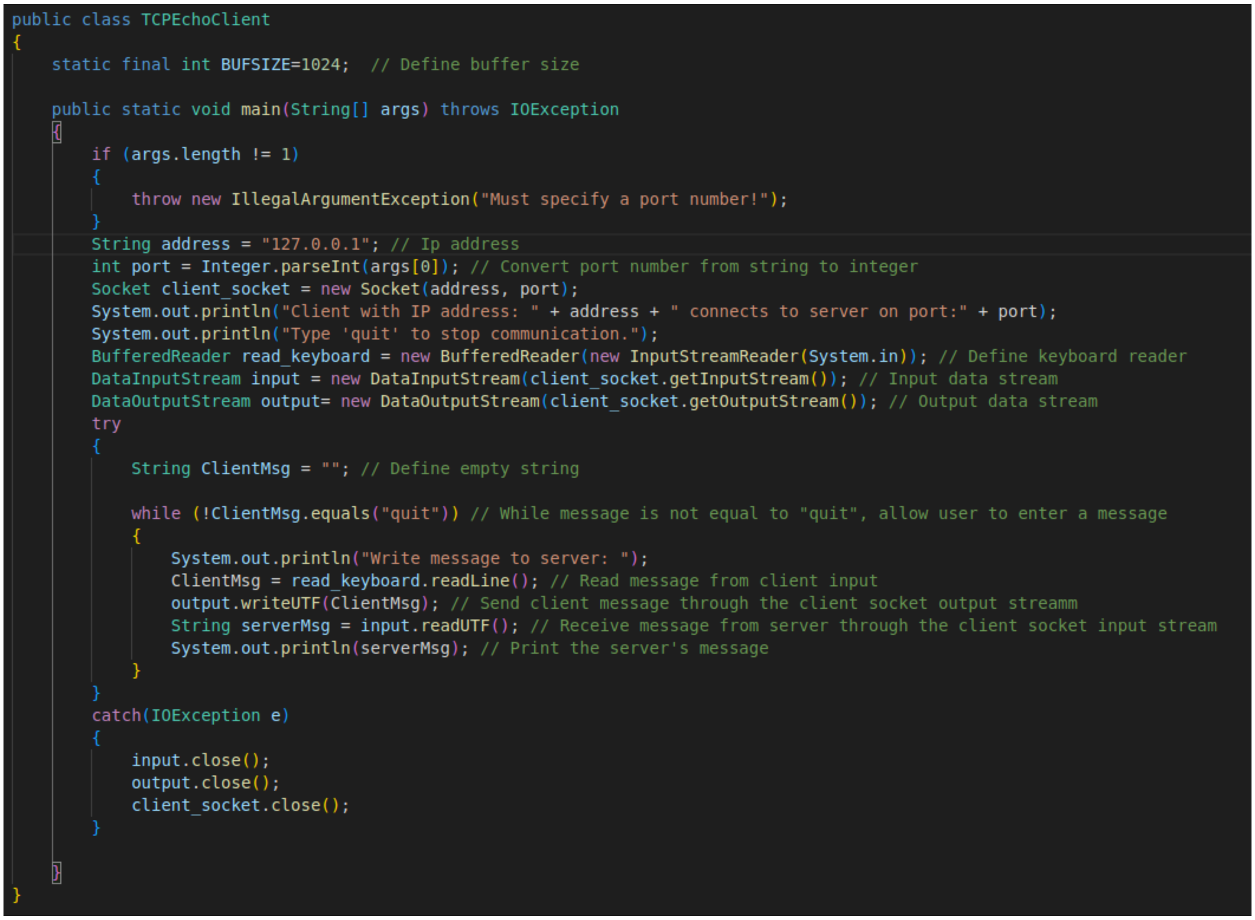
```
package nadir.Sockets;
import java.net.*;  // need this for InetAddress, Socket, ServerSocket
import java.io.*;   // need this for I/O stuff


public class TCPEchoServer {
    static final int BUFSIZE=1024; // define buffer size
    static public void main(String args[]) { // Main function

        if (args.length != 1){
            throw new IllegalArgumentException("Must specify a port number!");
        }

        try {
            String address = "127.0.0.1"; // Ip address
            int port = Integer.parseInt(args[0]); // Convert port number from string to integer
            ServerSocket ss = new ServerSocket(port);

            while (true) {
                System.out.println("The server is listening on port:" + port);
                Socket s = ss.accept();
                System.out.println("Client connexion is successful");
                handleClient(s);
            }

        } catch (IOException e) {
            System.out.println("Fatal I/O Error !");
            System.exit(0);

        }

    }
```



```
static void handleClient(Socket s) throws IOException
{
    String clientaddress = s.getInetAddress().getHostName();
    System.out.println("Connection from client: " + clientaddress); // Print client address
    DataInputStream input = new DataInputStream(s.getInputStream());
    DataOutputStream output=new DataOutputStream(s.getOutputStream());
    String ClientMsg = "";


    try
    {
        while (!ClientMsg.equals("quit"))
        {

                ClientMsg = input.readUTF();
                System.out.println("Client says: " + ClientMsg);
                String ServerMsg = "Server replies back: " + ClientMsg + "\n";
                output.writeUTF(ServerMsg);
        }
        System.out.println("Closing connection with client \n");

    }   catch(IOException e)
    {
        input.close();
        output.close();
        s.close();
    }
}
```

2.1.2. TCPEchoClient class

- Define buffer size
- Make sure port number is specified
- Convert port number to int
- Instantiate a client socket and a buffer reader
- Create the socket's input and output data streams
- Read the client message from the buffer reader input
- Send the client message to server through the output stream
- Get server's reply message from through the client socket input stream

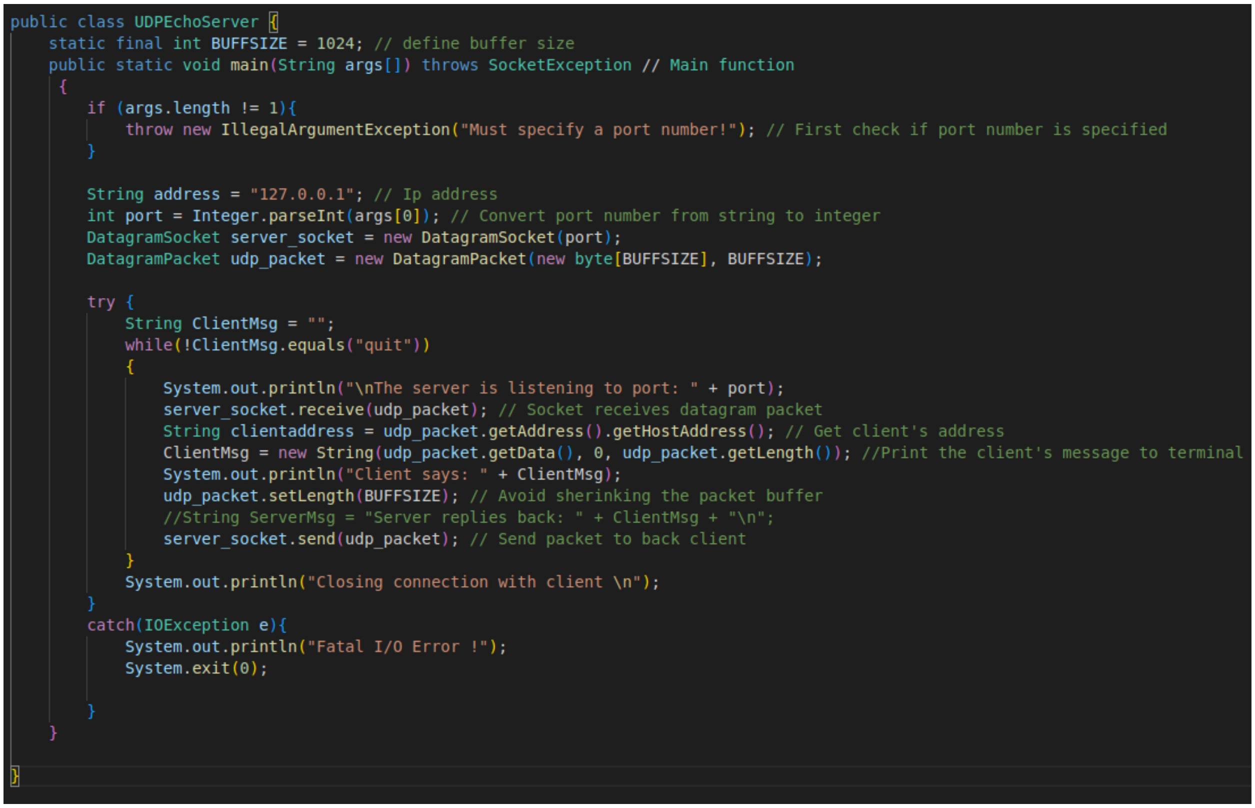


```java
public class TCPEchoClient
{
    static final int BUFSIZE=1024;  // Define buffer size

    public static void main(String[] args) throws IOException
    {
        if (args.length != 1)
        {
            throw new IllegalArgumentException("Must specify a port number!");
        }
        String address = "127.0.0.1"; // Ip address
        int port = Integer.parseInt(args[0]); // Convert port number from string to integer
        Socket client_socket = new Socket(address, port);
        System.out.println("Client with IP address: " + address + " connects to server on port:" + port);
        System.out.println("Type 'quit' to stop communication.");
        BufferedReader read_keyboard = new BufferedReader(new InputStreamReader(System.in)); // Define keyboard reader
        DataInputStream input = new DataInputStream(client_socket.getInputStream()); // Input data stream
        DataOutputStream output= new DataOutputStream(client_socket.getOutputStream()); // Output data stream
        try
        {
            String ClientMsg = ""; // Define empty string

            while (!ClientMsg.equals("quit")) // While message is not equal to "quit", allow user to enter a message
            {
                System.out.println("Write message to server: ");
                ClientMsg = read_keyboard.readLine(); // Read message from client input
                output.writeUTF(ClientMsg); // Send client message through the client socket output streamm
                String serverMsg = input.readUTF(); // Receive message from server through the client socket input stream
                System.out.println(serverMsg); // Print the server's message
            }
        }
        catch(IOException e)
        {
            input.close();
            output.close();
            client_socket.close();
        }

    }
}
```

2.2. UDP Communication

2.2.1. UDPEchoServer class

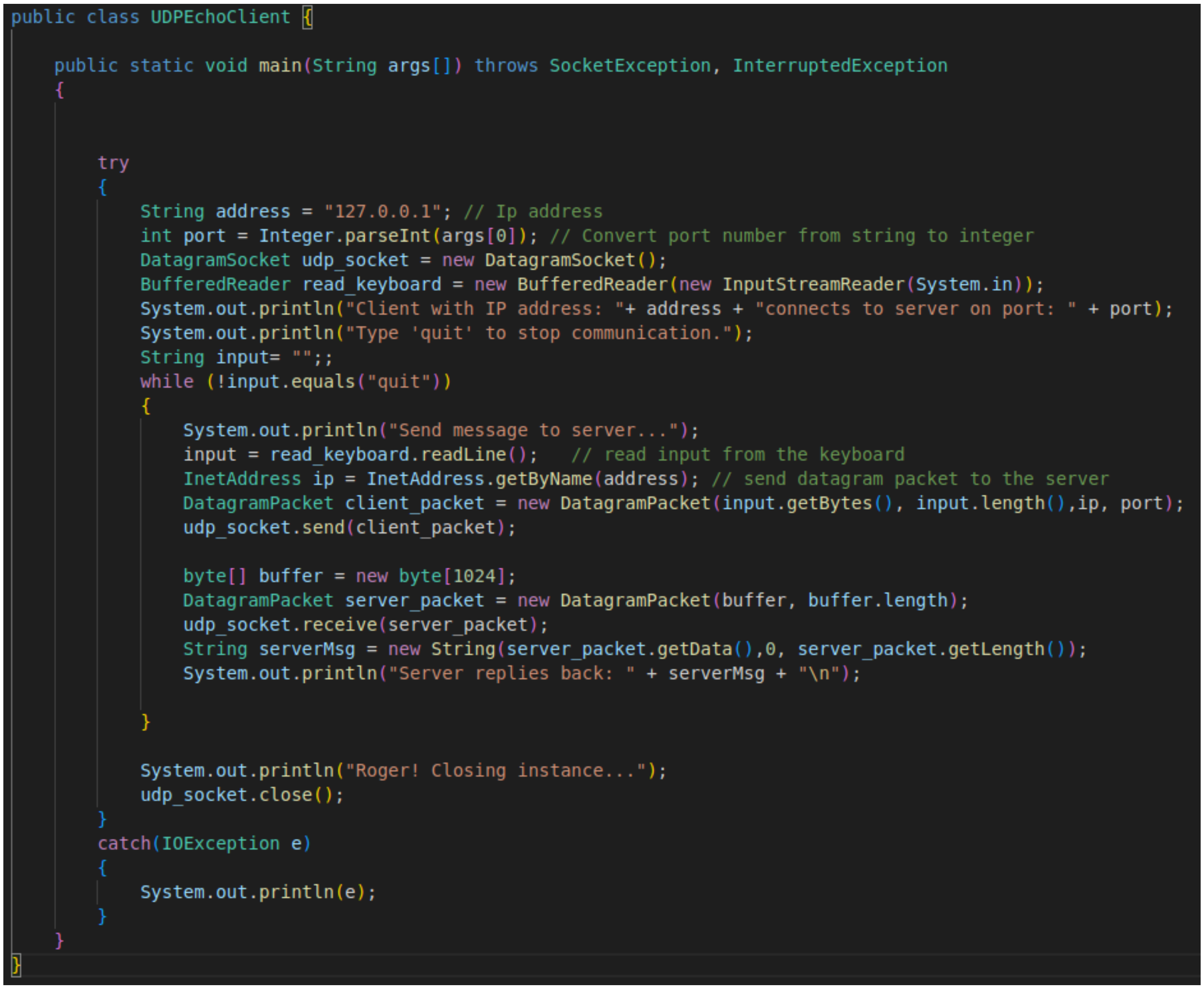
- Server socket is created using a DatagramSocket class
- UDP packet is received by the server socket
- The client’s address and payload are extracted from the received packet
- The server replies back by sending the same udp packet



```
public class UDPEchoServer {
    static final int BUFFSIZE = 1024; // define buffer size
    public static void main(String args[]) throws SocketException // Main function
     {
        if (args.length != 1){
            throw new IllegalArgumentException("Must specify a port number!"); // First check if port number is specified
        }

        String address = "127.0.0.1"; // Ip address
        int port = Integer.parseInt(args[0]); // Convert port number from string to integer
        DatagramSocket server_socket = new DatagramSocket(port);
        DatagramPacket udp_packet = new DatagramPacket(new byte[BUFFSIZE], BUFFSIZE);

        try {
            String ClientMsg = "";
            while(!ClientMsg.equals("quit"))
            {
                System.out.println("\nThe server is listening to port: " + port);
                server_socket.receive(udp_packet); // Socket receives datagram packet
                String clientaddress = udp_packet.getAddress().getHostAddress(); // Get client's address
                ClientMsg = new String(udp_packet.getData(), 0, udp_packet.getLength()); //Print the client's message to terminal
                System.out.println("Client says: " + ClientMsg);
                udp_packet.setLength(BUFFSIZE); // Avoid sherinking the packet buffer
                //String ServerMsg = "Server replies back: " + ClientMsg + "\n";
                server_socket.send(udp_packet); // Send packet to back client
            }
            System.out.println("Closing connection with client \n");
        }
        catch(IOException e){
            System.out.println("Fatal I/O Error !");
            System.exit(0);

        }
     }

}
```

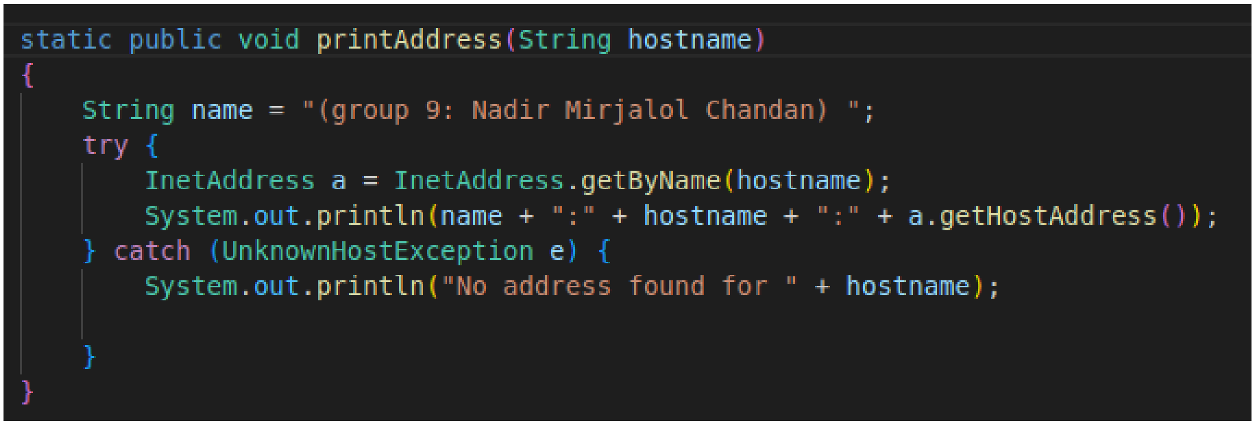
2.2.2. UDPEchoClient class

- Client udp socket is created with DatagramSocket class
- Instantiate a buffer reader object
- Read client input data and transform it into a DatagramPacket
- Use the client sock to send the udp packet
- Receive the reply back from server through the udp socket



```java
public class UDPEchoClient {

    public static void main(String args[]) throws SocketException, InterruptedException
    {


        try
        {
            String address = "127.0.0.1"; // Ip address
            int port = Integer.parseInt(args[0]); // Convert port number from string to integer
            DatagramSocket udp_socket = new DatagramSocket();
            BufferedReader read_keyboard = new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Client with IP address: "+ address + "connects to server on port: " + port);
            System.out.println("Type 'quit' to stop communication.");
            String input= "";;
            while (!input.equals("quit"))
            {
                System.out.println("Send message to server...");
                input = read_keyboard.readLine();   // read input from the keyboard
                InetAddress ip = InetAddress.getByName(address); // send datagram packet to the server
                DatagramPacket client_packet = new DatagramPacket(input.getBytes(), input.length(),ip, port);
                udp_socket.send(client_packet);

                byte[] buffer = new byte[1024];
                DatagramPacket server_packet = new DatagramPacket(buffer, buffer.length);
                udp_socket.receive(server_packet);
                String serverMsg = new String(server_packet.getData(),0, server_packet.getLength());
                System.out.println("Server replies back: " + serverMsg + "\n");

            }

            System.out.println("Roger! Closing instance...");
            udp_socket.close();
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
}
```

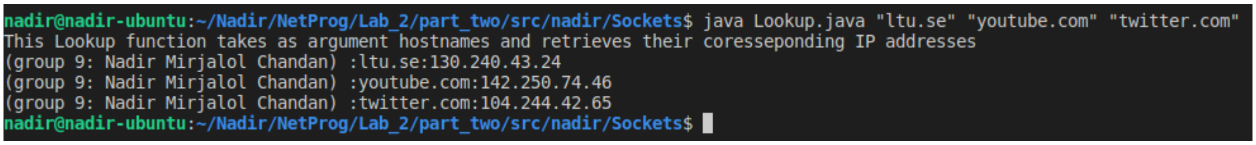
3. Modify the Lookup class so that it outputs our names in addition to the input parameters



```java
static public void printAddress(String hostname)
{
    String name = "(group 9: Nadir Mirjalol Chandan) ";
    try {
        InetAddress a = InetAddress.getByName(hostname);
        System.out.println(name + ":" + hostname + ":" + a.getHostAddress());
    } catch (UnknownHostException e) {
        System.out.println("No address found for " + hostname);

    }
}
```

- InetAddress: This class represents an Internet Protocol (IP) address.

- ○ String getByName(): determines the IP address of a host, given the host's name
- ○ String getHostAddress(): returns the IP address string in textual presentation

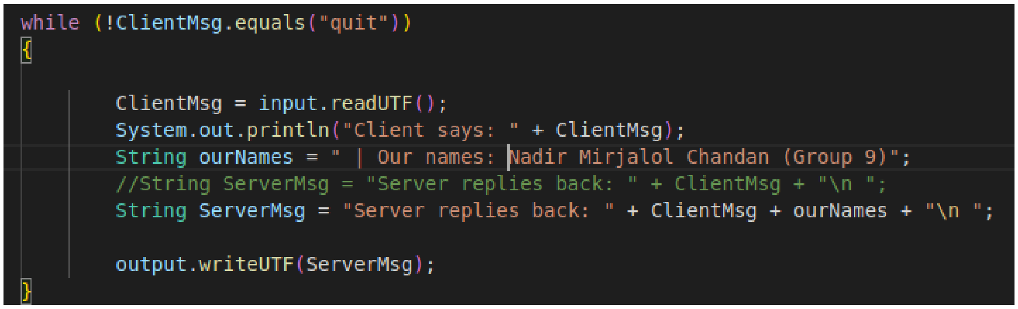


```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ java Lookup.java "ltu.se" "youtube.com" "twitter.com"
This Lookup function takes as argument hostnames and retrieves their coresseponding IP addresses
(group 9: Nadir Mirjalol Chandan) :ltu.se:130.240.43.24
(group 9: Nadir Mirjalol Chandan) :youtube.com:142.250.74.46
(group 9: Nadir Mirjalol Chandan) :twitter.com:104.244.42.65
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ 
```

In the command line, when running the Look up class, the input arguments are given to the main function as a list of strings “ltu.se” “youtube.com” “twitter.com”. The main function loops through all the values and performs a DNS lookup to retrieve the public IP address of these websites.

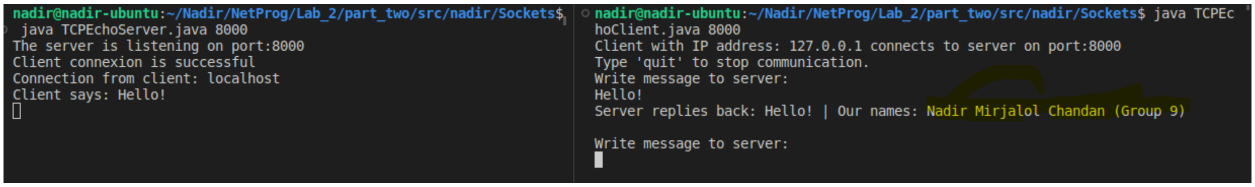
4. Modify both the TCPEchoServer and UDPEchoServer classes so that in addition to echoed input symbols the server would send back our names.

4.1. TCPEchoServer modification and outputs



```java
while (!ClientMsg.equals("quit"))
{

        ClientMsg = input.readUTF();
        System.out.println("Client says: " + ClientMsg);
        String ourNames = " | Our names: Nadir Mirjalol Chandan (Group 9)";
        //String ServerMsg = "Server replies back: " + ClientMsg + "\n ";
        String ServerMsg = "Server replies back: " + ClientMsg + ourNames + "\n ";

        output.writeUTF(ServerMsg);
}
```

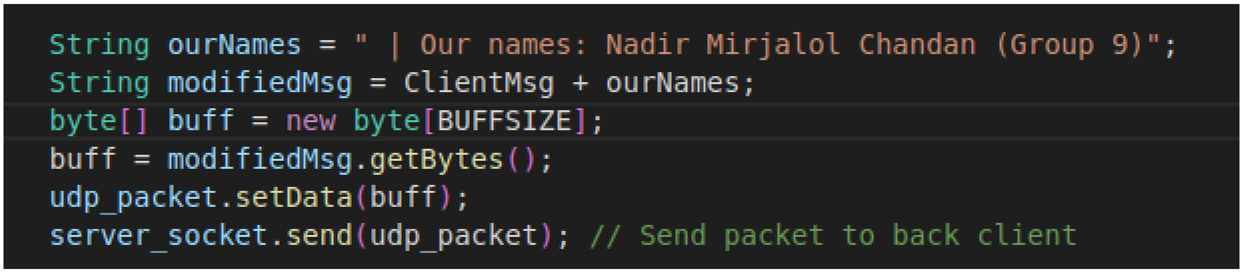
Server side **Client side**



```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$
 java TCPEchoServer.java 8000
The server is listening on port:8000
Client connexion is successful
Connection from client: localhost
Client says: Hello!
```

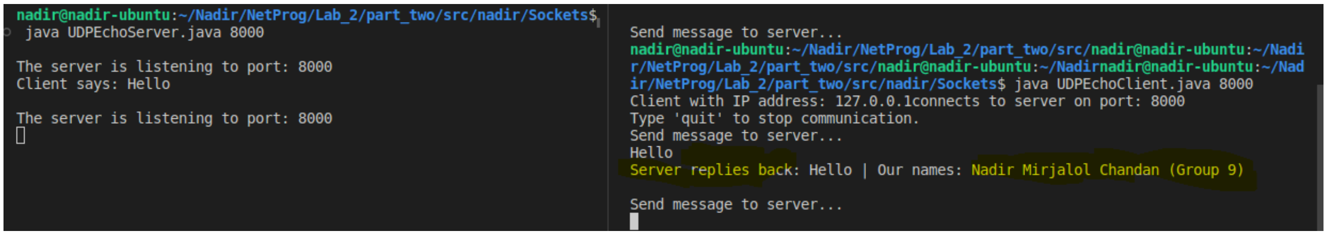
```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$ java TCPEc
hoClient.java 8000
Client with IP address: 127.0.0.1 connects to server on port:8000
Type 'quit' to stop communication.
Write message to server:
Hello!
Server replies back: Hello! | Our names: Nadir Mirjalol Chandan (Group 9)

Write message to server:
```

4.2. UDPEchoServer modification and outputs



```java
String ourNames = " | Our names: Nadir Mirjalol Chandan (Group 9)";
String modifiedMsg = ClientMsg + ourNames;
byte[] buff = new byte[BUFFSIZE];
buff = modifiedMsg.getBytes();
udp_packet.setData(buff);
server_socket.send(udp_packet); // Send packet to back client
```

Server side **Client side**



```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/Sockets$
 java UDPEchoServer.java 8000

The server is listening to port: 8000
Client says: Hello

The server is listening to port: 8000
```

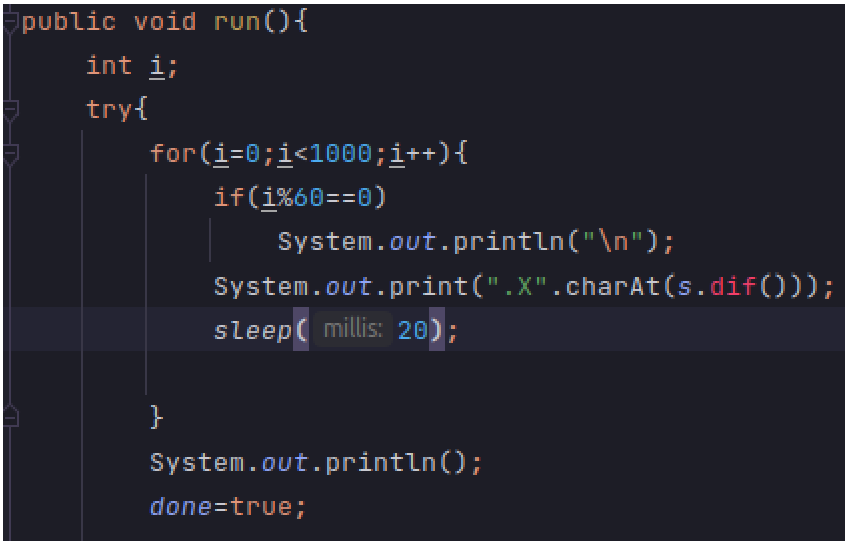
```
Send message to server...
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir@nadir-ubuntu:~/Nadi
r/NetProg/Lab_2/part_two/src/nadir@nadir-ubuntu:~/Nadirnadir@nadir-ubuntu:~/Nad
ir/NetProg/Lab_2/part_two/src/nadir/Sockets$ java UDPEchoClient.java 8000
Client with IP address: 127.0.0.1connects to server on port: 8000
Type 'quit' to stop communication.
Send message to server...
Hello
Server replies back: Hello | Our names: Nadir Mirjalol Chandan (Group 9)

Send message to server...
```

5. Compile the class Race0 in the threads part of the project.

5.1. What kind of behavior do you observe?

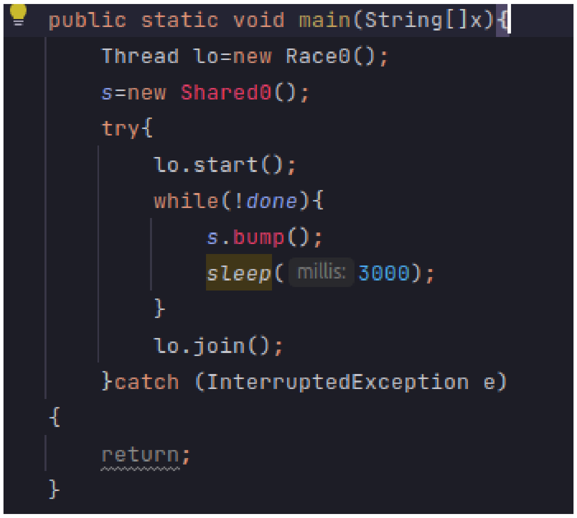
The main function of the Race0 class uses multithreading to run two functions simultaneously (defined in Shared), the first one (dif) that is started by the method lo.start(), which basically executes the run() defined in the Race0 thread. This method loops from i=0 to i=1000 and prints either a dot "." or "x" every 20ms. The returned value from diff(x, y) represents the index of string being printed (either . or X).

- If dif(x,y) = 1, print "."
- If dif(x,y) = 0, print "X"



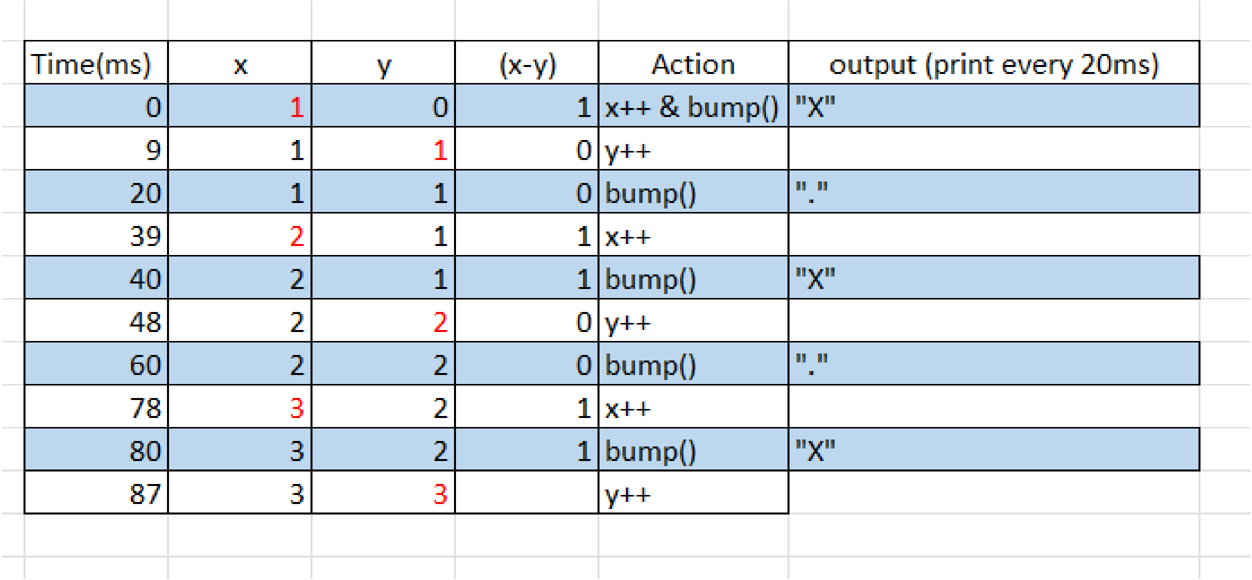
```java
public void run(){
    int i;
    try{
        for(i=0;i<1000;i++){
            if(i%60==0)
                System.out.println("\n");
            System.out.print(".X".charAt(s.dif()));
            sleep( millis: 20);

        }
        System.out.println();
        done=true;
```

Once the loop ends, the variable "done" is set to true. This variable controls the second function bump() called in the main function. While "done" is false, the bump method increments the variable x, then sleeps for 9ms, then increments y.



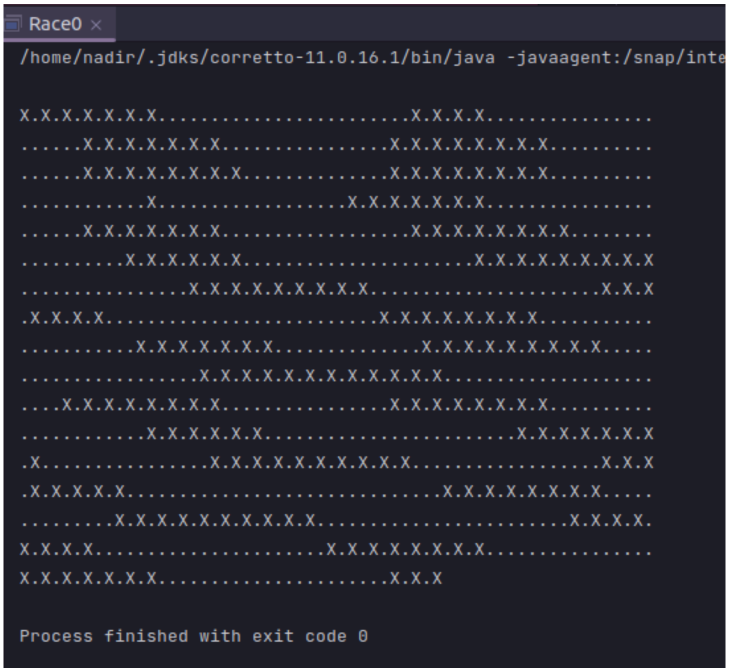
```
public static void main(String[]x){
    Thread lo=new Race0();
    s=new Shared0();
    try{
        lo.start();
        while(!done){
            s.bump();
            sleep( millis: 3000);
        }
        lo.join();
    }catch (InterruptedException e)
{
    return;
}
```

On top of that, the main function sleeps for 30 ms after executing bump, which means that x and y are incremented every 39 ms but shifted by 9ms.



Time(ms)	x	y	(x-y)	Action	output (print every 20ms)
0	1	0	1	x++ & bump()	"X"
9	1	1	0	y++	
20	1	1	0	bump()	"."
39	2	1	1	x++	
40	2	1	1	bump()	"X"
48	2	2	0	y++	
60	2	2	0	bump()	"."
78	3	2	1	x++	
80	3	2	1	bump()	"X"
87	3	3		y++	

The table above shows the variation of x y values and its according output during the first milliseconds. The red color indicates that the value has been incremented and it starts alternating at first. However, after a few iterations the output is observed to be printing only dots ".....".

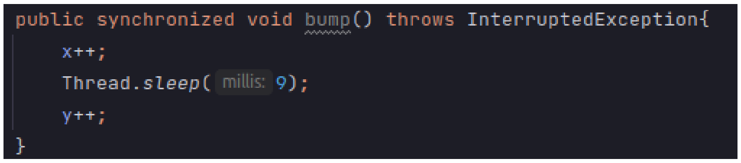


```
/home/nadir/.jdks/corretto-11.0.16.1/bin/java -javaagent:/snap/inte

X.X.X.X.X.X.X........................X.X.X.X................
......X.X.X.X.X.X.X................X.X.X.X.X.X.X.X..........
......X.X.X.X.X.X.X.X..............X.X.X.X.X.X.X.X..........
............X..................X.X.X.X.X.X.X................
......X.X.X.X.X.X.X..................X.X.X.X.X.X.X.X........
..........X.X.X.X.X.X......................X.X.X.X.X.X.X.X.X
................X.X.X.X.X.X.X.X.X......................X.X.X
.X.X.X.X..........................X.X.X.X.X.X.X.X...........
...........X.X.X.X.X.X.X..............X.X.X.X.X.X.X.X.X.....
.................X.X.X.X.X.X.X.X.X.X.X.X....................
....X.X.X.X.X.X.X.X................X.X.X.X.X.X.X.X..........
............X.X.X.X.X.X........................X.X.X.X.X.X.X
.X................X.X.X.X.X.X.X.X.X.X..................X.X.X
.X.X.X.X.X..............................X.X.X.X.X.X.X.X.....
.........X.X.X.X.X.X.X.X.X.X........................X.X.X.X.
X.X.X.X......................X.X.X.X.X.X.X.X................
X.X.X.X.X.X.X......................X.X.X

Process finished with exit code 0
```

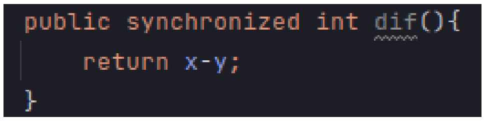
5.2. Why?

Since the variable X is shared by the two functions, when they both try to access the variable, an unpredictable behavior occurs in the value passed to both our function, which explains the asynchronous printing of “x” and “.”.

5.3. Make now both dif() and bump() methods synchronized. Compile the classes and run.



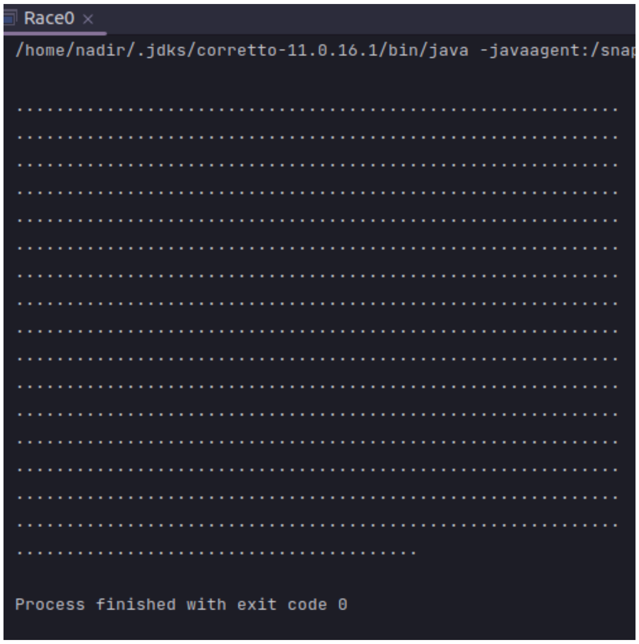
```
public synchronized void bump() throws InterruptedException{
    x++;
    Thread.sleep( millis: 9);
    y++;
}
```



```
public synchronized int dif(){
    return x-y;
}
```

5.4. How has the behavior changed now?

The output shows that in all iterations, only the character “.” has been printed, which means that dif(x,y)=0, therefore in all iterations variable x is equal to y. This means that the dif() method is executed before/after the incrementation of both x and y. Therefore, the two methods are executed one after the other, so they can not be run in parallel.



```
/home/nadir/.jdks/corretto-11.0.16.1/bin/java -javaagent:/snap

............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
............................................................
........................................

Process finished with exit code 0
```

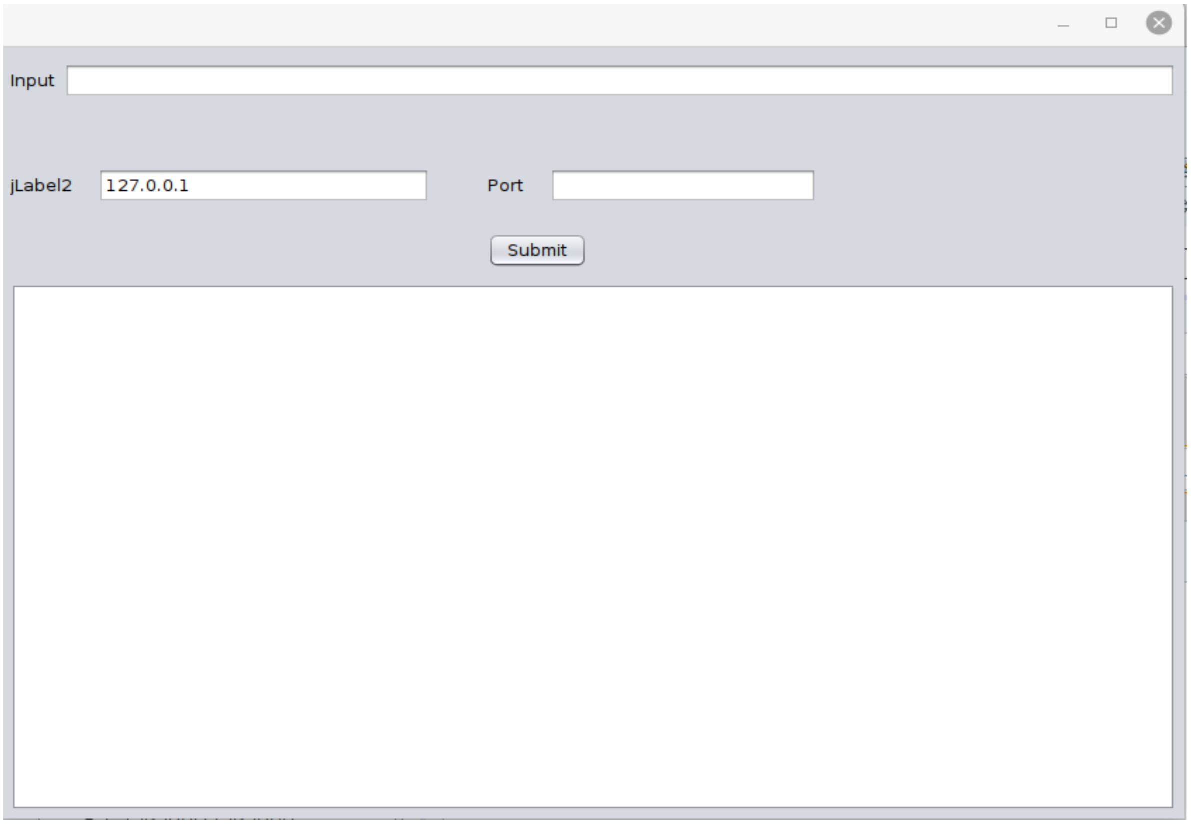
Part Three: Client Server Application

This part is the extension of the part 1 of the lab. Here we need to integrate client and server architecture to the application. The GUI works as the client side application and the The entered commands is executed and run as the server-side application.To create this client-server architecture we use the multi threaded TCP connection. From the gui front , two extra fields need to be added i.e. one for the ip address of the server and other for the port of the server. The final GUI is shown in the below picture.

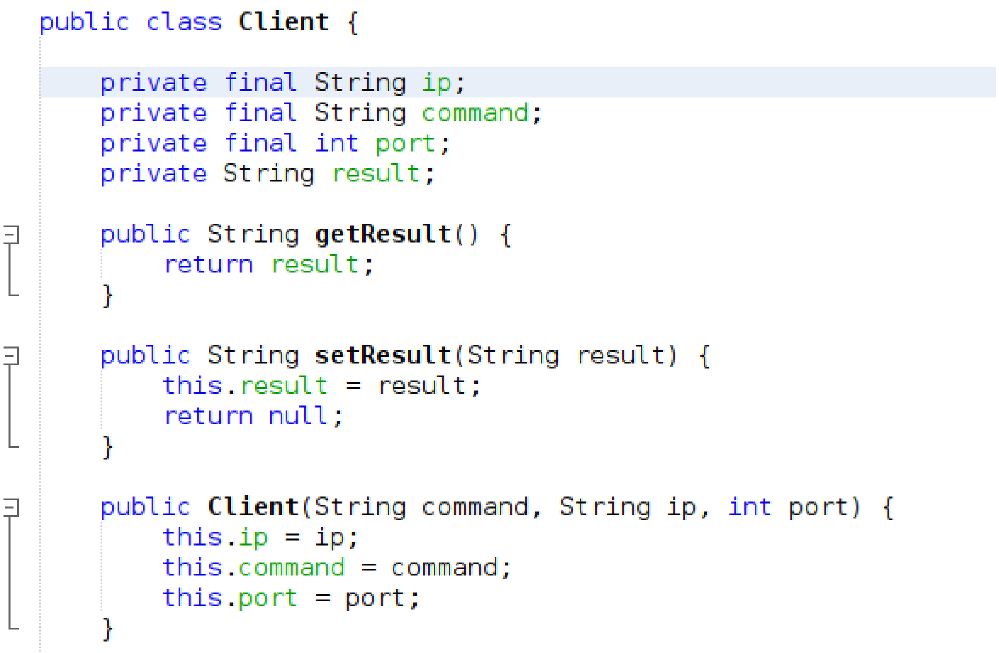
To implement the client and the server we create one class for the client and two classes for the server in two separate java file.

Client-Side Program: A client can communicate with a server using this code. This involves

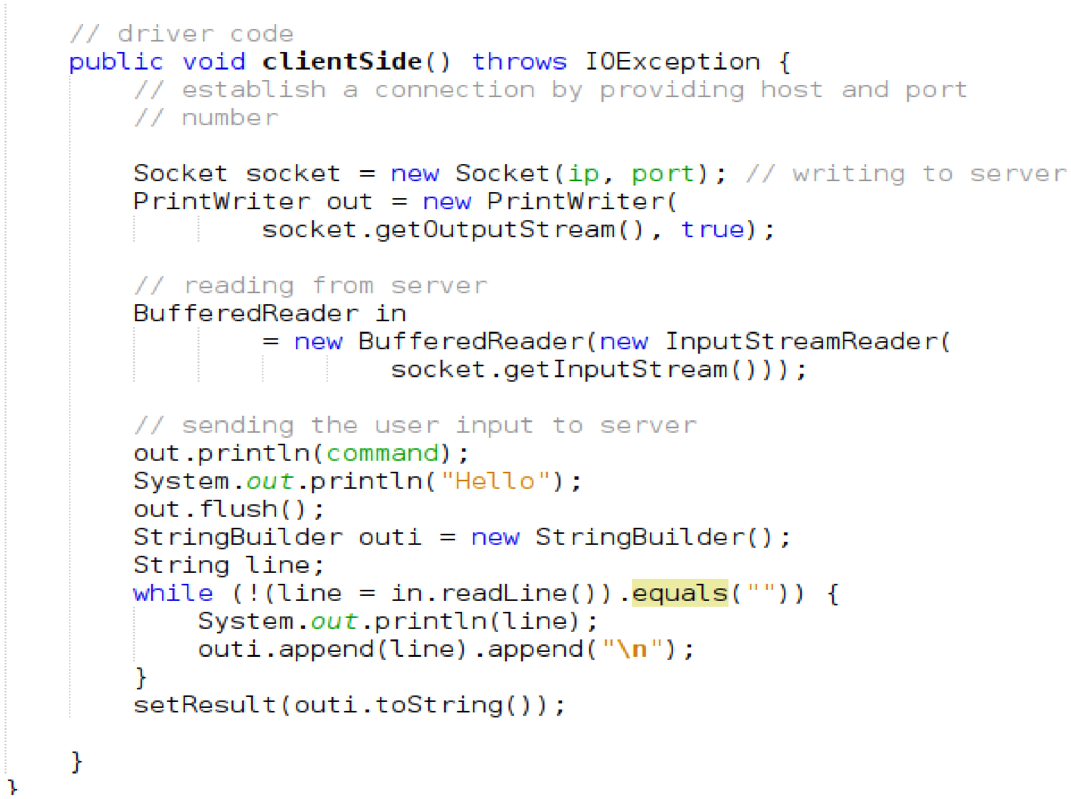
1. **Establish a Socket Connection**
2. **Communication**



127.0.0.1



```
public class Client {

    private final String ip;
    private final String command;
    private final int port;
    private String result;

    public String getResult() {
        return result;
    }

    public String setResult(String result) {
        this.result = result;
        return null;
    }

    public Client(String command, String ip, int port) {
        this.ip = ip;
        this.command = command;
        this.port = port;
    }
```



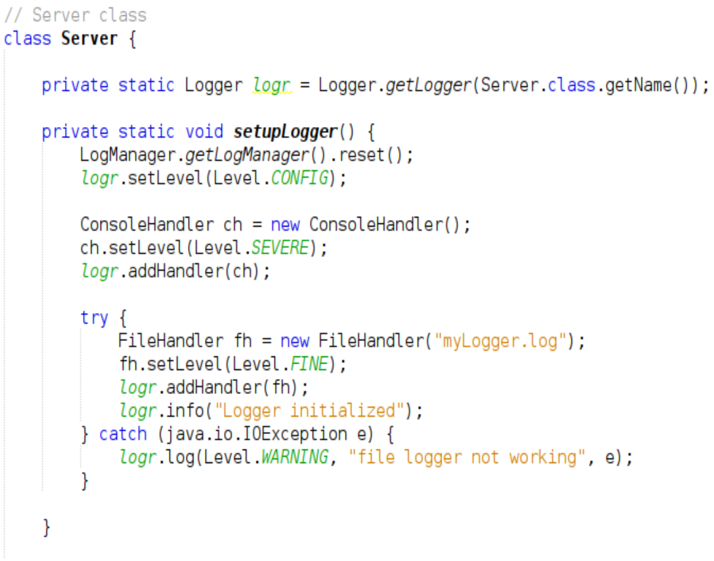
```
    // driver code
    public void clientSide() throws IOException {
        // establish a connection by providing host and port
        // number

        Socket socket = new Socket(ip, port); // writing to server
        PrintWriter out = new PrintWriter(
                socket.getOutputStream(), true);

        // reading from server
        BufferedReader in
                = new BufferedReader(new InputStreamReader(
                        socket.getInputStream()));

        // sending the user input to server
        out.println(command);
        System.out.println("Hello");
        out.flush();
        StringBuilder outi = new StringBuilder();
        String line;
        while (!(line = in.readLine()).equals("")) {
            System.out.println(line);
            outi.append(line).append("\n");
        }
        setResult(outi.toString());

    }
}
```

For the server side we have two class

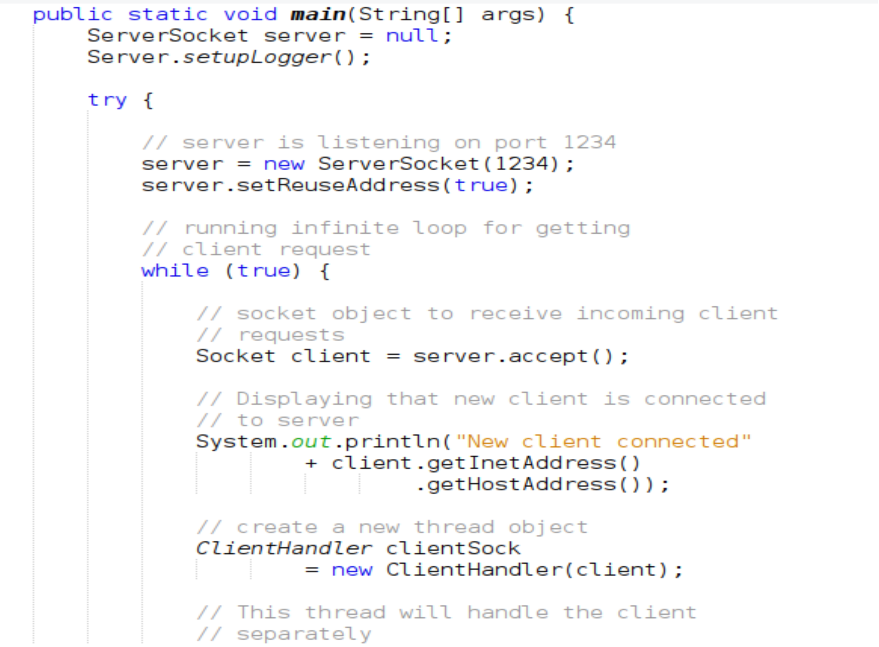
Server class: Here we create the thread object. The steps involved are

1. **Establish the Connection**: Server socket object inside a while loop continuously accept the connection from the client
2. **Obtaining the streams:** The socket object for the current requests is used to extract the inputstream object and outputstream object.
3. **Creating an Handler Object**: new handler object is created with the port number and with the streams obtained
4. **Invoking the start() method**: start method is invoked.

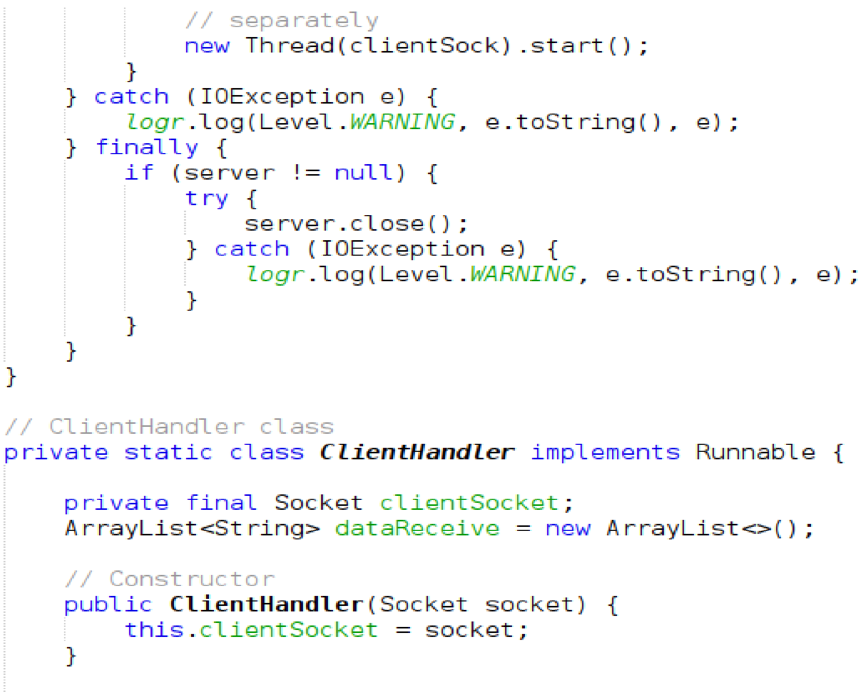
clientHandle class: Here the thread is created implementing the runnable interface and inside the run method it reads the clients messages.



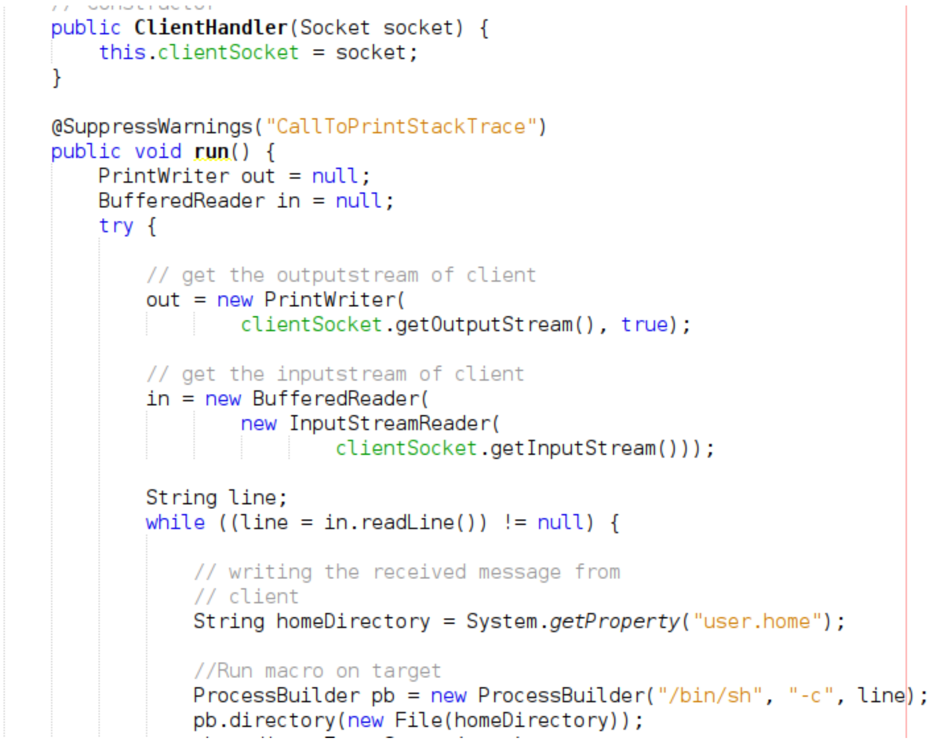
```
// Server class
class Server {

    private static Logger logr = Logger.getLogger(Server.class.getName());

    private static void setupLogger() {
        LogManager.getLogManager().reset();
        logr.setLevel(Level.CONFIG);

        ConsoleHandler ch = new ConsoleHandler();
        ch.setLevel(Level.SEVERE);
        logr.addHandler(ch);

        try {
            FileHandler fh = new FileHandler("myLogger.log");
            fh.setLevel(Level.FINE);
            logr.addHandler(fh);
            logr.info("Logger initialized");
        } catch (java.io.IOException e) {
            logr.log(Level.WARNING, "file logger not working", e);
        }

    }
```



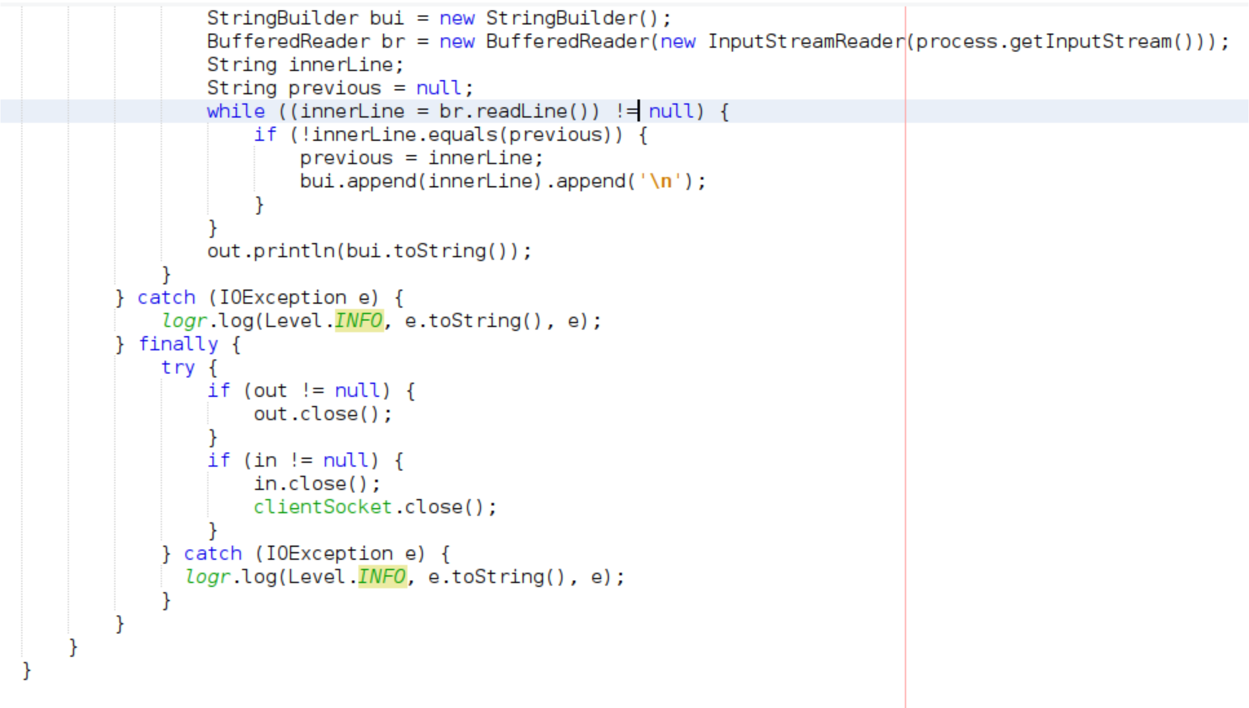
```
public static void main(String[] args) {
    ServerSocket server = null;
    Server.setupLogger();

    try {

        // server is listening on port 1234
        server = new ServerSocket(1234);
        server.setReuseAddress(true);

        // running infinite loop for getting
        // client request
        while (true) {

            // socket object to receive incoming client
            // requests
            Socket client = server.accept();

            // Displaying that new client is connected
            // to server
            System.out.println("New client connected"
                    + client.getInetAddress()
                            .getHostAddress());

            // create a new thread object
            ClientHandler clientSock
                    = new ClientHandler(client);

            // This thread will handle the client
            // separately
```



```
            // separately
            new Thread(clientSock).start();
        }
    } catch (IOException e) {
        logr.log(Level.WARNING, e.toString(), e);
    } finally {
        if (server != null) {
            try {
                server.close();
            } catch (IOException e) {
                logr.log(Level.WARNING, e.toString(), e);
            }
        }
    }
}

// ClientHandler class
private static class ClientHandler implements Runnable {

    private final Socket clientSocket;
    ArrayList<String> dataReceive = new ArrayList<>();

    // Constructor
    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }
```



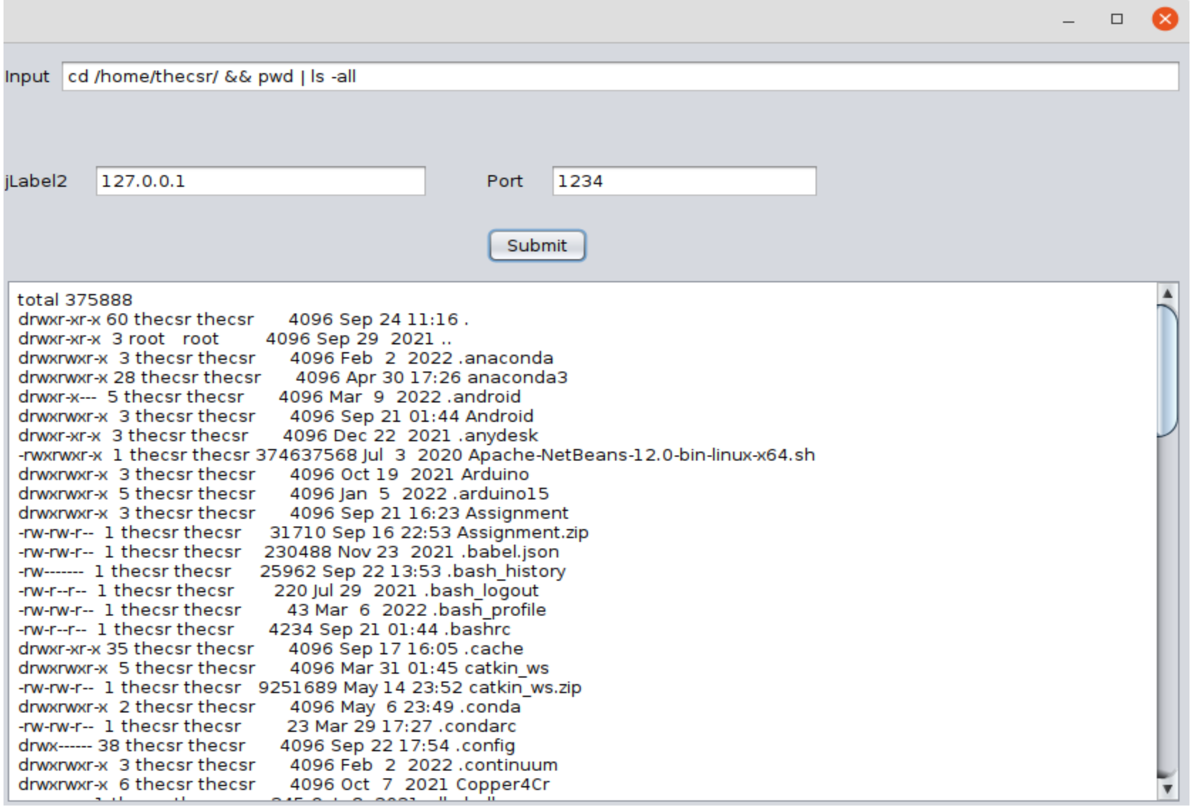
```
    // Constructor
    public ClientHandler(Socket socket) {
        this.clientSocket = socket;
    }

    @SuppressWarnings("CallToPrintStackTrace")
    public void run() {
        PrintWriter out = null;
        BufferedReader in = null;
        try {

            // get the outputstream of client
            out = new PrintWriter(
                    clientSocket.getOutputStream(), true);

            // get the inputstream of client
            in = new BufferedReader(
                    new InputStreamReader(
                            clientSocket.getInputStream()));

            String line;
            while ((line = in.readLine()) != null) {

                // writing the received message from
                // client
                String homeDirectory = System.getProperty("user.home");

                //Run macro on target
                ProcessBuilder pb = new ProcessBuilder("/bin/sh", "-c", line);
                pb.directory(new File(homeDirectory));
```



```
                StringBuilder bui = new StringBuilder();
                BufferedReader br = new BufferedReader(new InputStreamReader(process.getInputStream()));
                String innerLine;
                String previous = null;
                while ((innerLine = br.readLine()) != null) {
                    if (!innerLine.equals(previous)) {
                        previous = innerLine;
                        bui.append(innerLine).append('\n');
                    }
                }
                out.println(bui.toString());
            }
        } catch (IOException e) {
            logr.log(Level.INFO, e.toString(), e);
        } finally {
            try {
                if (out != null) {
                    out.close();
                }
                if (in != null) {
                    in.close();
                    clientSocket.close();
                }
            } catch (IOException e) {
              logr.log(Level.INFO, e.toString(), e);
            }
        }
    }
}
```

To add the logging capability to the server-side application, we have created a new method called setUp logger and have added the functionality to log both on the termina and in the file. The logger file is called myLogger.log and is located in the same direvtory as the java codes. We can added logging levels like warning, info , severe according to the situation in the code.

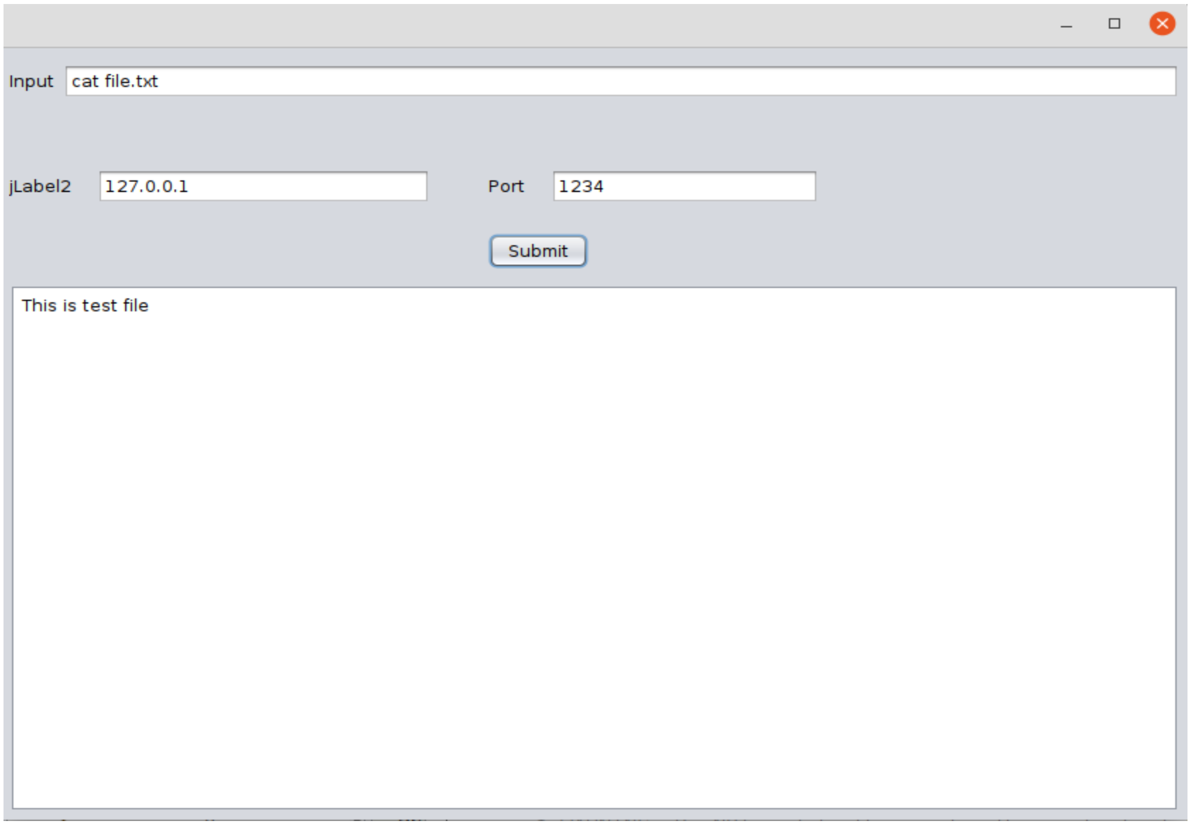
The command is executed from the gui, which is executed on the server side application. The screenshot of the command executed is given below.



Input cd /home/thecsr/ && pwd | ls -all

jLabel2 127.0.0.1 Port 1234

```
total 375888
drwxr-xr-x 60 thecsr thecsr      4096 Sep 24 11:16 .
drwxr-xr-x  3 root   root        4096 Sep 29  2021 ..
drwxrwxr-x  3 thecsr thecsr      4096 Feb  2  2022 .anaconda
drwxrwxr-x 28 thecsr thecsr      4096 Apr 30 17:26 anaconda3
drwxr-x---  5 thecsr thecsr      4096 Mar  9  2022 .android
drwxrwxr-x  3 thecsr thecsr      4096 Sep 21 01:44 Android
drwxr-xr-x  3 thecsr thecsr      4096 Dec 22  2021 .anydesk
-rwxrwxr-x  1 thecsr thecsr 374637568 Jul  3  2020 Apache-NetBeans-12.0-bin-linux-x64.sh
drwxrwxr-x  3 thecsr thecsr      4096 Oct 19  2021 Arduino
drwxrwxr-x  5 thecsr thecsr      4096 Jan  5  2022 .arduino15
drwxrwxr-x  3 thecsr thecsr      4096 Sep 21 16:23 Assignment
-rw-rw-r--  1 thecsr thecsr     31710 Sep 16 22:53 Assignment.zip
-rw-rw-r--  1 thecsr thecsr    230488 Nov 23  2021 .babel.json
-rw-------  1 thecsr thecsr     25962 Sep 22 13:53 .bash_history
-rw-r--r--  1 thecsr thecsr       220 Jul 29  2021 .bash_logout
-rw-rw-r--  1 thecsr thecsr        43 Mar  6  2022 .bash_profile
-rw-r--r--  1 thecsr thecsr      4234 Sep 21 01:44 .bashrc
drwxr-xr-x 35 thecsr thecsr      4096 Sep 17 16:05 .cache
drwxrwxr-x  5 thecsr thecsr      4096 Mar 31 01:45 catkin_ws
-rw-rw-r--  1 thecsr thecsr   9251689 May 14 23:52 catkin_ws.zip
drwxrwxr-x  2 thecsr thecsr      4096 May  6 23:49 .conda
-rw-rw-r--  1 thecsr thecsr        23 Mar 29 17:27 .condarc
drwx------ 38 thecsr thecsr      4096 Sep 22 17:54 .config
drwxrwxr-x  3 thecsr thecsr      4096 Feb  2  2022 .continuum
drwxrwxr-x  6 thecsr thecsr      4096 Oct  7  2021 Copper4Cr
```

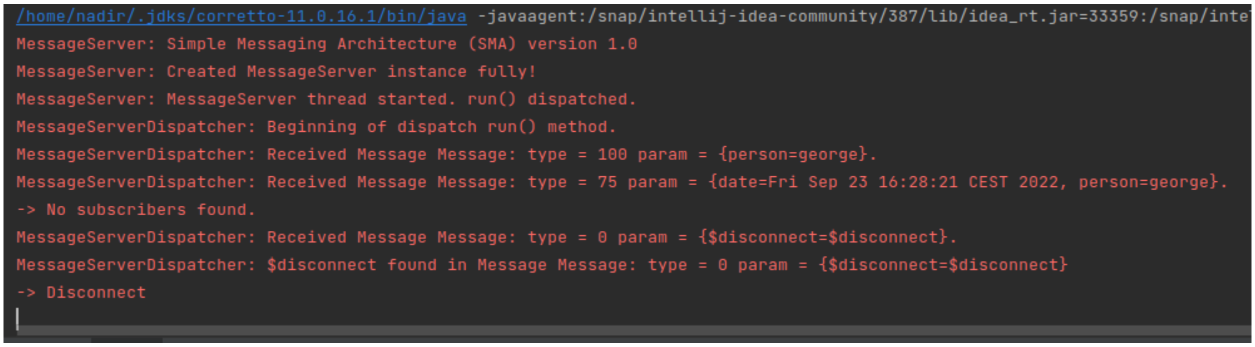


Input cat file.txt

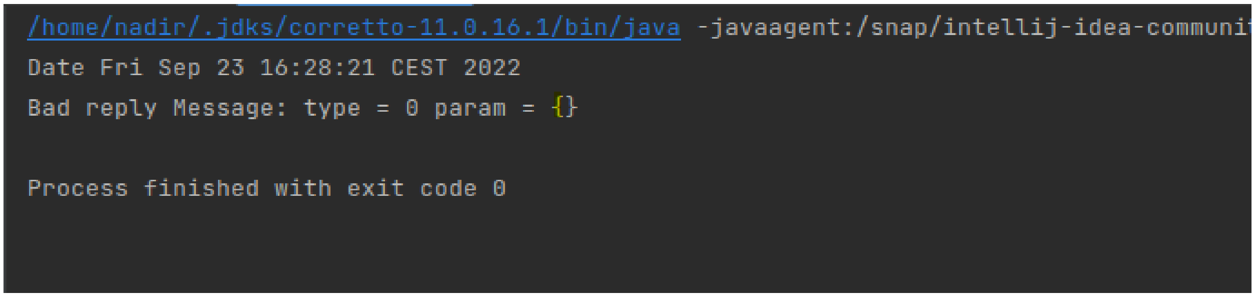
jLabel2 127.0.0.1 Port 1234

```
This is test file
```

Part Four: Simple Messaging Architecture



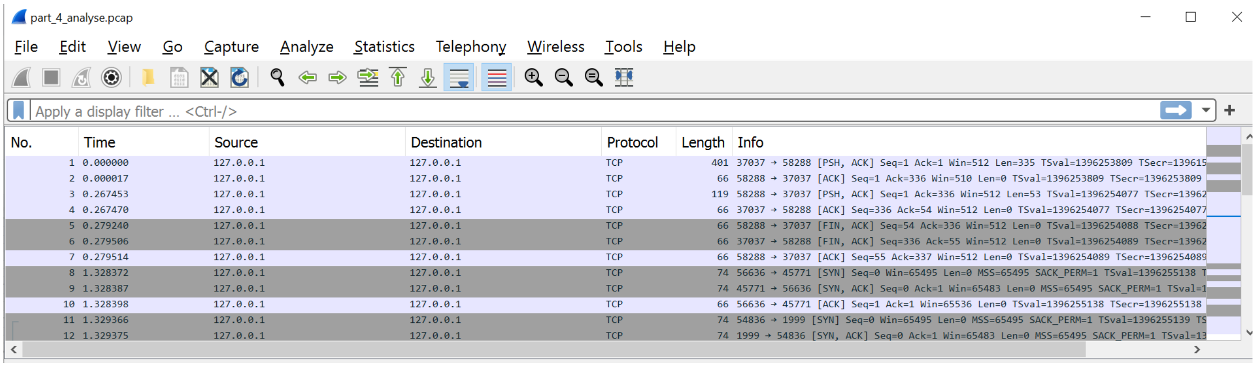
```
/home/nadir/.jdks/corretto-11.0.16.1/bin/java -javaagent:/snap/intellij-idea-community/387/lib/idea_rt.jar=33359:/snap/inte
MessageServer: Simple Messaging Architecture (SMA) version 1.0
MessageServer: Created MessageServer instance fully!
MessageServer: MessageServer thread started. run() dispatched.
MessageServerDispatcher: Beginning of dispatch run() method.
MessageServerDispatcher: Received Message Message: type = 100 param = {person=george}.
MessageServerDispatcher: Received Message Message: type = 75 param = {date=Fri Sep 23 16:28:21 CEST 2022, person=george}.
-> No subscribers found.
MessageServerDispatcher: Received Message Message: type = 0 param = {$disconnect=$disconnect}.
MessageServerDispatcher: $disconnect found in Message Message: type = 0 param = {$disconnect=$disconnect}
-> Disconnect
```



```
/home/nadir/.jdks/corretto-11.0.16.1/bin/java -javaagent:/snap/intellij-idea-communi
Date Fri Sep 23 16:28:21 CEST 2022
Bad reply Message: type = 0 param = {}

Process finished with exit code 0
```

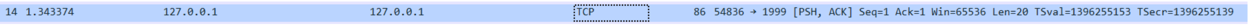
1. Analyzing Wireshark results

The DateServer class is run on port 1999 and the DateClient class is connected to it. We display the captured communication using Wireshark.

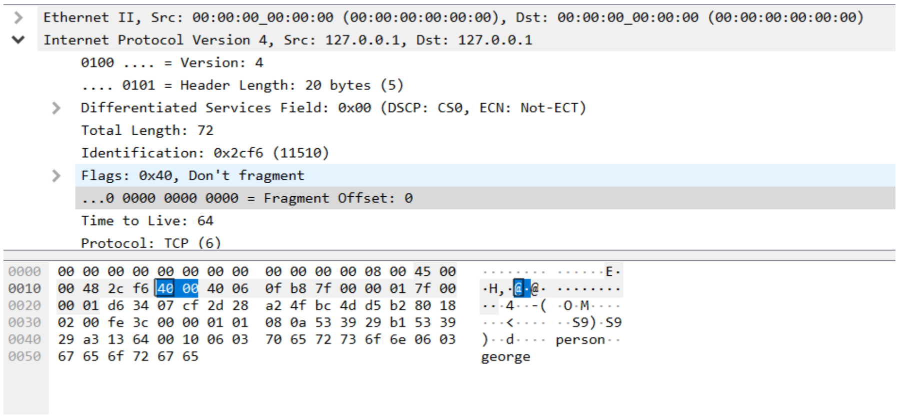

part_4_analyse.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	401	37037 → 58288 [PSH, ACK] Seq=1 Ack=1 Win=512 Len=335 TSval=1396253809 TSecr=139615
2	0.000017	127.0.0.1	127.0.0.1	TCP	66	58288 → 37037 [ACK] Seq=1 Ack=336 Win=510 Len=0 TSval=1396253809 TSecr=1396253809
3	0.267453	127.0.0.1	127.0.0.1	TCP	119	58288 → 37037 [PSH, ACK] Seq=1 Ack=336 Win=512 Len=53 TSval=1396254077 TSecr=13962
4	0.267470	127.0.0.1	127.0.0.1	TCP	66	37037 → 58288 [ACK] Seq=336 Ack=54 Win=512 Len=0 TSval=1396254077 TSecr=1396254077
5	0.279240	127.0.0.1	127.0.0.1	TCP	66	58288 → 37037 [FIN, ACK] Seq=54 Ack=336 Win=512 Len=0 TSval=1396254088 TSecr=13962
6	0.279506	127.0.0.1	127.0.0.1	TCP	66	37037 → 58288 [FIN, ACK] Seq=336 Ack=55 Win=512 Len=0 TSval=1396254089 TSecr=13962
7	0.279514	127.0.0.1	127.0.0.1	TCP	66	58288 → 37037 [ACK] Seq=55 Ack=337 Win=512 Len=0 TSval=1396254089 TSecr=1396254089
8	1.328372	127.0.0.1	127.0.0.1	TCP	74	56636 → 45771 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1396255138 T
9	1.328387	127.0.0.1	127.0.0.1	TCP	74	45771 → 56636 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1
10	1.328398	127.0.0.1	127.0.0.1	TCP	66	56636 → 45771 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1396255138 TSecr=1396255138
11	1.329366	127.0.0.1	127.0.0.1	TCP	74	54836 → 1999 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1396255139 TS
12	1.329375	127.0.0.1	127.0.0.1	TCP	74	1999 → 54836 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=13

- Communication starts by a request from the client to connect
- Server acknowledges the connection
- Client sends a message containing the key value pair (person, george)

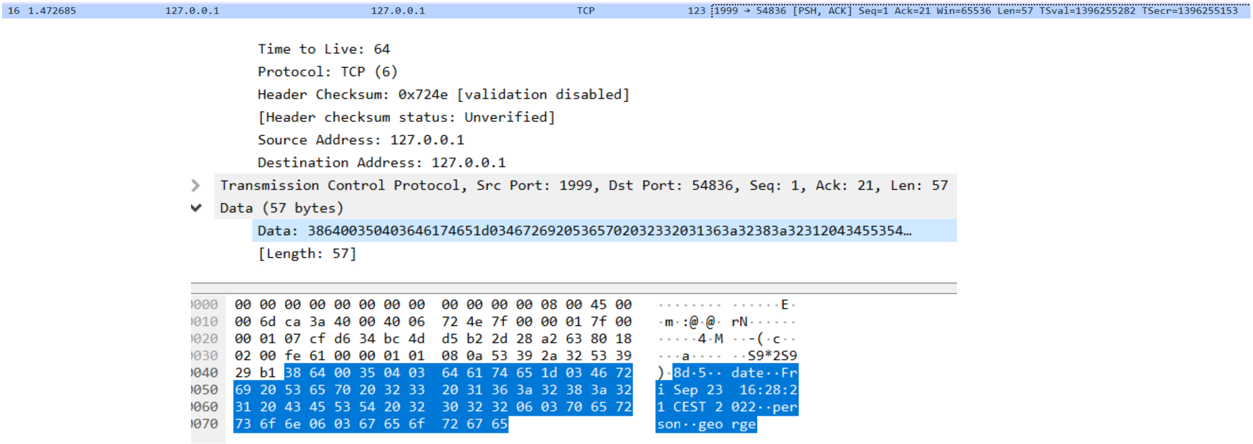


```
14 1.343374    127.0.0.1    127.0.0.1    TCP    86  54836 → 1999 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20 TSval=1396255153 TSecr=1396255139
```



```
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
v Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
     0100 .... = Version: 4
     .... 0101 = Header Length: 20 bytes (5)
   > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
     Total Length: 72
     Identification: 0x2cf6 (11510)
   > Flags: 0x40, Don't fragment
     ...0 0000 0000 0000 = Fragment Offset: 0
     Time to Live: 64
     Protocol: TCP (6)

0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
0010  00 48 2c f6 40 00 40 06  0f b8 7f 00 00 01 7f 00   ·H,·@·@· ········
0020  00 01 d6 34 07 cf 2d 28  a2 4f bc 4d d5 b2 80 18   ···4··-( ·O·M····
0030  02 00 fe 3c 00 00 01 01  08 0a 53 39 29 b1 53 39   ···<···· ··S9)·S9
0040  29 a3 13 64 00 10 06 03  70 65 72 73 6f 6e 06 03   )··d···· person··
0050  67 65 6f 72 67 65                                  george
```

- The server acknowledges the message, then it replies back to the client with a message containing the previous payload (person, george) in addition to the current date

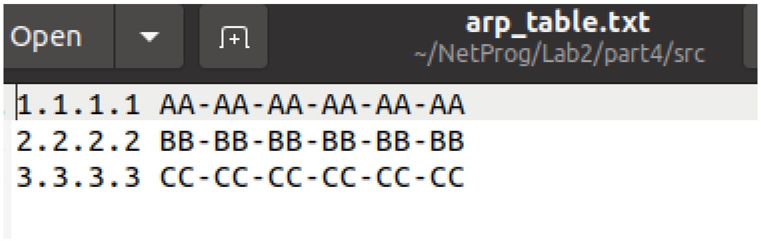


```
16 1.472685     127.0.0.1     127.0.0.1     TCP     123 1999 → 54836 [PSH, ACK] Seq=1 Ack=21 Win=65536 Len=57 TSval=1396255282 TSecr=1396255153

     Time to Live: 64
     Protocol: TCP (6)
     Header Checksum: 0x724e [validation disabled]
     [Header checksum status: Unverified]
     Source Address: 127.0.0.1
     Destination Address: 127.0.0.1
> Transmission Control Protocol, Src Port: 1999, Dst Port: 54836, Seq: 1, Ack: 21, Len: 57
v Data (57 bytes)
     Data: 386400350403646174651d03467269205365702032332031363a32383a32312043455354…
     [Length: 57]

)000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ········ ······E·
)010  00 6d ca 3a 40 00 40 06  72 4e 7f 00 00 01 7f 00   ·m·:@·@· rN······
)020  00 01 07 cf d6 34 bc 4d  d5 b2 2d 28 a2 63 80 18   ·····4·M ··-(·c··
)030  02 00 fe 61 00 00 01 01  08 0a 53 39 2a 32 53 39   ···a···· ··S9*2S9
)040  29 b1 38 64 00 35 04 03  64 61 74 65 1d 03 46 72   )·8d·5·· date··Fr
)050  69 20 53 65 70 20 32 33  20 31 36 3a 32 38 3a 32   i Sep 23  16:28:2
)060  31 20 43 45 53 54 20 32  30 32 32 06 03 70 65 72   1 CEST 2 022··per
)070  73 6f 6e 06 03 67 65 6f  72 67 65                  son··geo rge
```

- Finally, the connection is closed by the client and the server acknowledges.

2. Extending the architecture of SMA:

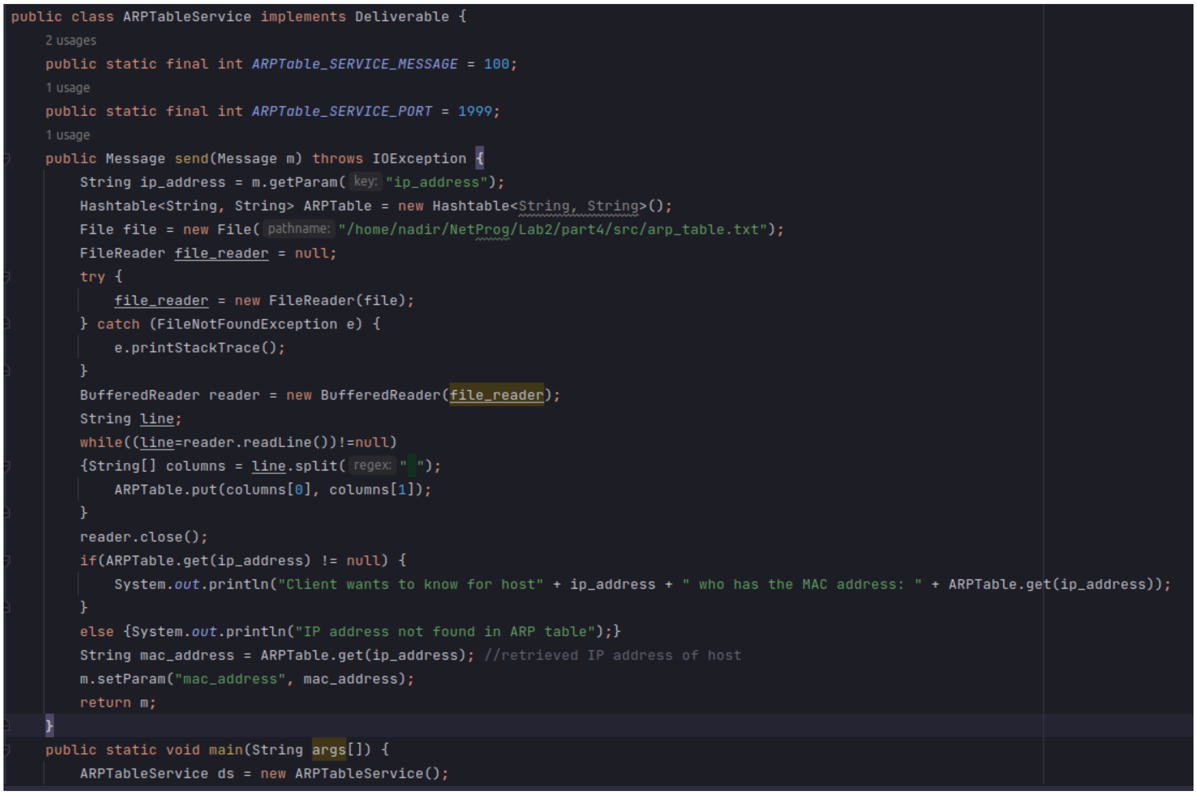
- Instead of creating a DNS service, we thought about creating an ARP table service which return mac address when given an IP as input
- The arp_table.txt file contains ip addresses mapped to mac addresses



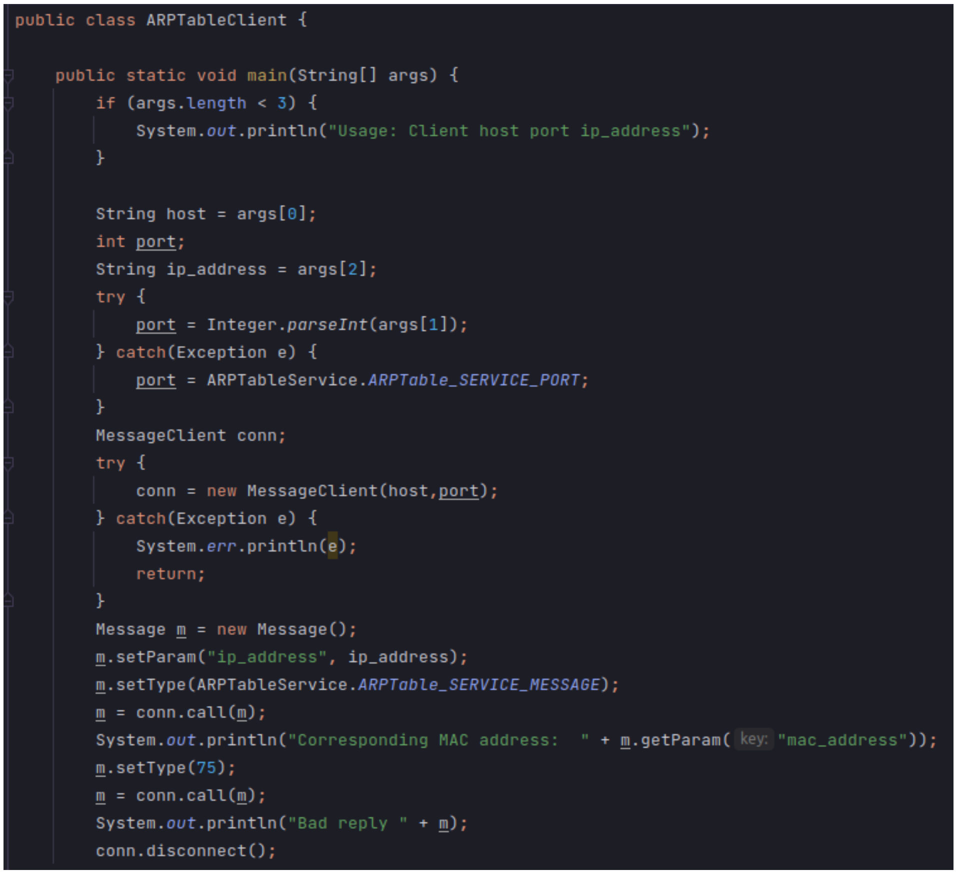
```
1.1.1.1 AA-AA-AA-AA-AA-AA
2.2.2.2 BB-BB-BB-BB-BB-BB
3.3.3.3 CC-CC-CC-CC-CC-CC
```

- ARPTableService class
    - We use a hashtable to read content in the arp_table.txt file
    - We read the file by using a buffer raeder
    - The targeted ip address is provided by the client as an input
    - The server then uses the ip address provided to look up for the mac address in the hashtable
    - Once the mac address retrieved, we set the parameter “mac_address” to its value, within the message object (m)

- ARPTableClient class
    - Within the message (m), we set a parameter “ip_address” and its corresponding value comes from input Args[2]
    - The class use this value to

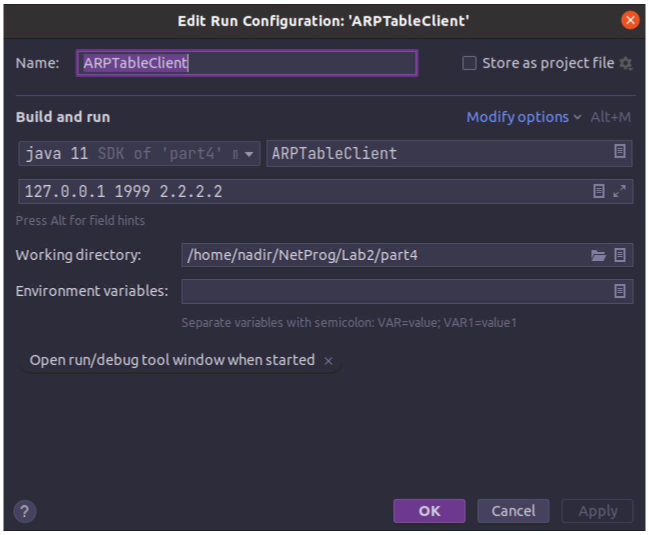


```
public class ARPTableService implements Deliverable {
    2 usages
    public static final int ARPTable_SERVICE_MESSAGE = 100;
    1 usage
    public static final int ARPTable_SERVICE_PORT = 1999;
    1 usage
    public Message send(Message m) throws IOException {
        String ip_address = m.getParam( key: "ip_address");
        Hashtable<String, String> ARPTable = new Hashtable<String, String>();
        File file = new File( pathname: "/home/nadir/NetProg/Lab2/part4/src/arp_table.txt");
        FileReader file_reader = null;
        try {
            file_reader = new FileReader(file);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        BufferedReader reader = new BufferedReader(file_reader);
        String line;
        while((line=reader.readLine())!=null)
        {String[] columns = line.split( regex: " ");
            ARPTable.put(columns[0], columns[1]);
        }
        reader.close();
        if(ARPTable.get(ip_address) != null) {
            System.out.println("Client wants to know for host" + ip_address + " who has the MAC address: " + ARPTable.get(ip_address));
        }
        else {System.out.println("IP address not found in ARP table");}
        String mac_address = ARPTable.get(ip_address); //retrieved IP address of host
        m.setParam("mac_address", mac_address);
        return m;
    }
    public static void main(String args[]) {
        ARPTableService ds = new ARPTableService();
```



```java
public class ARPTableClient {

    public static void main(String[] args) {
        if (args.length < 3) {
            System.out.println("Usage: Client host port ip_address");
        }

        String host = args[0];
        int port;
        String ip_address = args[2];
        try {
            port = Integer.parseInt(args[1]);
        } catch(Exception e) {
            port = ARPTableService.ARPTable_SERVICE_PORT;
        }
        MessageClient conn;
        try {
            conn = new MessageClient(host,port);
        } catch(Exception e) {
            System.err.println(e);
            return;
        }
        Message m = new Message();
        m.setParam("ip_address", ip_address);
        m.setType(ARPTableService.ARPTable_SERVICE_MESSAGE);
        m = conn.call(m);
        System.out.println("Corresponding MAC address:  " + m.getParam( key: "mac_address"));
        m.setType(75);
        m = conn.call(m);
        System.out.println("Bad reply " + m);
        conn.disconnect();
```

- We test the client by giving the following arguments
  - Args[0] = 127.0.0.1 (server address)
  - Args[1] = 1999 (port number)
  - Args[2] = IP address to map
- The server responds back with the Mac address BB-BB-BB-BB-BB-BB



Edit Run Configuration: 'ARPTableClient'

Name: ARPTableClient

□ Store as project file

Build and run

java 11 SDK of 'part4' | ARPTableClient

127.0.0.1 1999 2.2.2.2

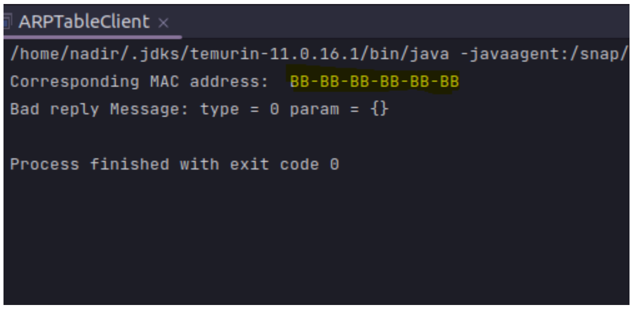
Press Alt for field hints

Working directory: /home/nadir/NetProg/Lab2/part4

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started ×



ARPTableClient ×

```
/home/nadir/.jdks/temurin-11.0.16.1/bin/java -javaagent:/snap/
Corresponding MAC address:  BB-BB-BB-BB-BB-BB
Bad reply Message: type = 0 param = {}

Process finished with exit code 0
```

Part Five: Java Netprog Patterns - Security

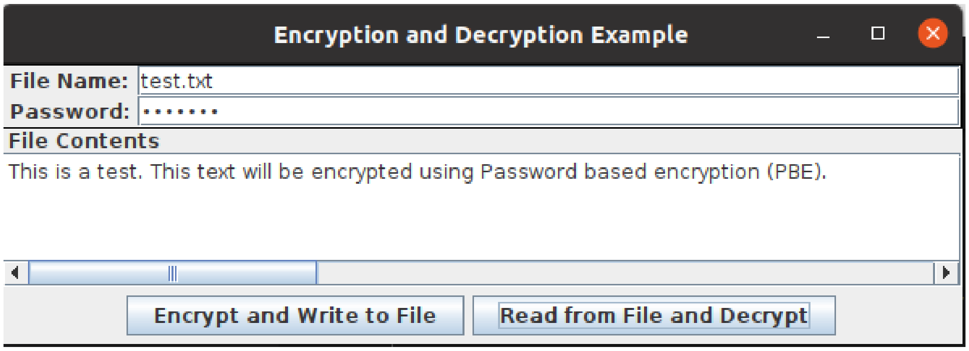
1. **Java Cryptography Extension (JCE)**
   JCA is a complementary framework added on top of the Java platform to provide a set of implementations for encryption, key generation, key agreement, as well as other security features such as asymmetric encryption (RSA) and Password-based encryption (PBE) etc..

   Com.sun packages cannot be accessed without explicitly setting them while compiling the class. We solve the issue by running the code like this:

```
nadir@nadir-ubuntu:~/Nadir/NetProg/Lab_2/part_two/src/nadir/security/jce$ java
--add-exports java.base/com.sun.crypto.provider=ALL-UNNAMED
EncipherDecipher.java
```

- The EncipherDecipher class provides a Password-based Encryption (PBE)
- A file is created for testing and “group9” was set as a password
- The written content is then encrypted as demonstrated in the figures below
- The file can be decrypted back to its original format using the box “Read from file and Decrypt”.

Before encryption



Encryption and Decryption Example
Encrypt and Write to File
Read from File and Decrypt

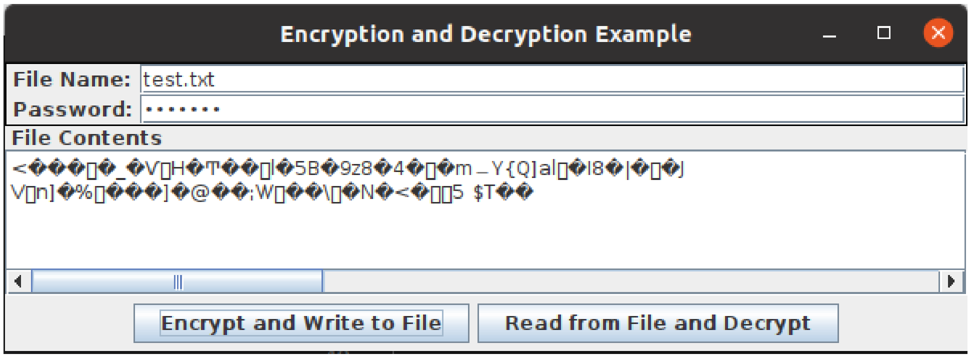
File Name: test.txt

Password: •••••••

File Contents

This is a test. This text will be encrypted using Password based encryption (PBE).

After encryption


Encryption and Decryption Example
Encrypt and Write to File
Read from File and Decrypt

File Name: test.txt
Password: •••••••

File Contents

```
<����▯�_�V▯H�T��▯l�5B�9z8�4�▯�m_Y{Q]al▯�l8�|�▯�J
V▯n]�%▯����]�@��;W▯��\▯�N�<�▯▯5 $T��
```

2. **Java Secure Socket Extension (JSSE)**

Highly sensitive and confidential data such as passwords, credit card numbers etc… must be encrypted whilst traveling across a network, particularly the internet. One has to guarantee data confidentiality by making it unreadable to unauthorized parties, as well as making sure its integrity is protected. No modification either intentional or unintentional should take place. Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols help protect privacy and integrity of data while transmission.

JSSE provides an API framework based on these protocols, hence abstracting the complex security algorithms and handshake mechanisms, which minimizes the risk of having dangerous security vulnerabilities. This API simplifies application design and serves as a building block for developers when implementing security features. JSSE extends java networking socket classes and includes several functionalities for data encryption, message integrity and both server and client authentication.

In this part of the lab, an authentication of the client occurs at the server's side and JSSE is used to secure communication sockets between the two entities.

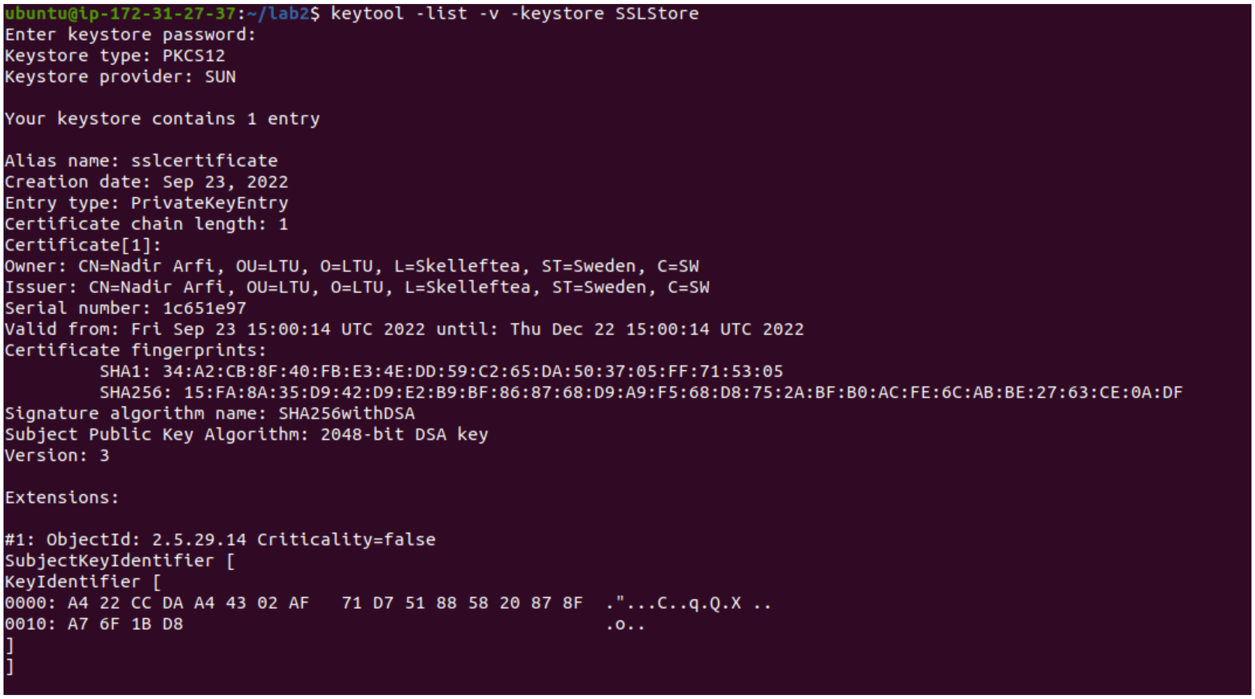
2.1. Login Server:

- First, we launch an EC2 instance on AWS and we configure a security group to allow all TCP inbound traffic in order to allow for SSL connection, then we install java jdk-11 on the machine.
- The login server file is copied using SCP from our local machine to the EC2 instance

```
sudo scp -i KEY_D7001D_gr9.pem /home/nadir/NetProg/Lab2/part5/src/LoginServer.java
ubuntu@13.48.70.157:~/lab2/
```

- We then generate the self-signed certificate and place it in the KeyStore

```
keytool -genkeypair -alias sslcertificate -keystore SSLStore
```

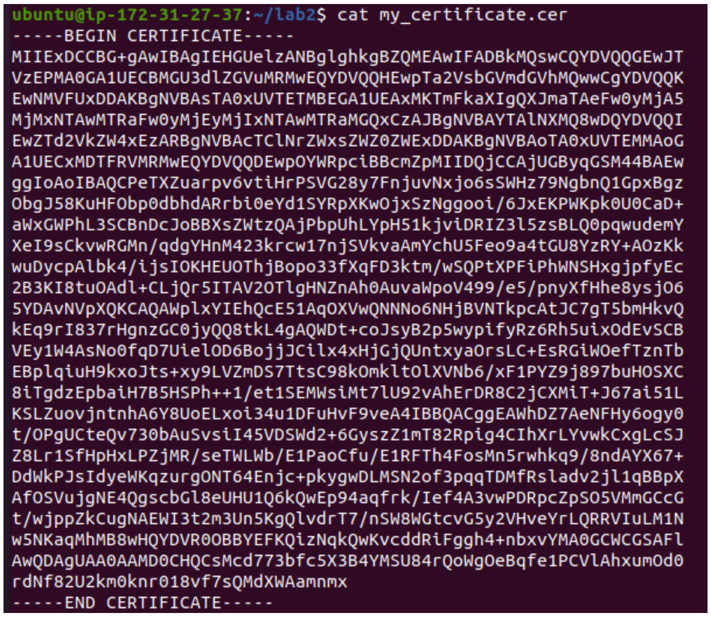


```
ubuntu@ip-172-31-27-37:~/lab2$ keytool -list -v -keystore SSLStore
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: sslcertificate
Creation date: Sep 23, 2022
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
Issuer: CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
Serial number: 1c651e97
Valid from: Fri Sep 23 15:00:14 UTC 2022 until: Thu Dec 22 15:00:14 UTC 2022
Certificate fingerprints:
         SHA1: 34:A2:CB:8F:40:FB:E3:4E:DD:59:C2:65:DA:50:37:05:FF:71:53:05
         SHA256: 15:FA:8A:35:D9:42:D9:E2:B9:BF:86:87:68:D9:A9:F5:68:D8:75:2A:BF:B0:AC:FE:6C:AB:BE:27:63:CE:0A:DF
Signature algorithm name: SHA256withDSA
Subject Public Key Algorithm: 2048-bit DSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: A4 22 CC DA A4 43 02 AF   71 D7 51 88 58 20 87 8F  ."...C..q.Q.X ..
0010: A7 6F 1B D8                                        .o..
]
]
```

- We export the SSL certificate

```
keytool -export -alias sslcertificate -keystore SSLStore -rfc -file my_certificate.cer
```

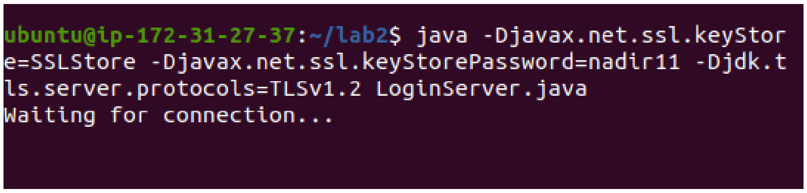
- The generated SSL certificate



```
ubuntu@ip-172-31-27-37:~/lab2$ cat my_certificate.cer
-----BEGIN CERTIFICATE-----
MIIExDCCBG+gAwIBAgIEHGUelzANBglghkgBZQMEAwIFADBkMQswCQYDVQQGEwJT
VzEPMA0GA1UECBMGU3dlZGVuMRMwEQYDVQQHEwpTa2VsbGVmdGVhMQwwCgYDVQQK
EwNMVFUxDDAKBgNVBAsTA0xUVTETMBEGA1UEAxMKTmFkaXIgQXJmaTAeFw0yMjA5
MjMxNTAwMTRaFw0yMjEyMjIxNTAwMTRaMGQxCzAJBgNVBAYTAlNXMQ8wDQYDVQQI
EwZTd2VkZW4xEzARBgNVBAcTClNrZWxsZWZ0ZWExDDAKBgNVBAoTA0xUVTEMMAoG
A1UECxMDTFRVMRMwEQYDVQQDEwpOYWRpciBBcmZpMIIDQjCCAjUGByqGSM44BAEw
ggIoAoIBAQCPeTXZuarpv6vtiHrPSVG28y7FnjuvNxjo6sSWHz79NgbnQ1GpxBgz
ObgJ58KuHFObp0dbhdARrbi0eYd1SYRpXKwOjxSzNggooi/6JxEKPWKpk0U0CaD+
aWxGWPhL3SCBnDcJoBBXsZWtzQAjPbpUhLYpH51kjviDRIZ3l5zsBLQ0pqwudemY
XeI9sCkvwRGMn/qdgYHnM423krcw17njSVkvaAmYchU5Feo9a4tGU8YzRY+AOzKk
wuDycpAlbk4/ijsIOKHEUOThjBopo33fXqFD3ktm/wSQPtXPFiPhWNSHxgjpfyEc
2B3KI8tuOAdl+CLjQr5ITAV2OTlgHNZnAh0AuvaWpoV499/e5/pnyXfHhe8ysjO6
5YDAvNVpXQKCAQAWplxYIEhQcE51AqOXVwQNNNo6NHjBVNTkpcAtJC7gT5bmHkvQ
kEq9rI837rHgnzGC0jyQQ8tkL4gAQWDt+coJsyB2p5wypifyRz6Rh5uixOdEvSCB
VEy1W4AsNo0fqD7UielOD6BojjJCilx4xHjGjQUntxyaOrsLC+EsRGiWOefTznTb
EBplqiuH9kxoJts+xy9LVZmDS7TtsC98kOmkltOlXVNb6/xF1PYZ9j897buHOSXC
8iTgdzEpbaiH7B5HSPh++1/et1SEMWsiMt7lU92vAhErDR8C2jCXMiT+J67ai51L
KSLZuovjntnhA6Y8UoELxoi34u1DFuHvF9veA4IBBQACggEAWhDZ7AeNFHy6ogy0
t/OPgUCteQv730bAuSvsiI45VDSWd2+6GyszZ1mT82Rpig4CIhXrLYvwkCxgLcSJ
Z8Lr1SfHpHxLPZjMR/seTWLWb/E1PaoCfu/E1RFTh4FosMn5rwhkq9/8ndAYX67+
DdWkPJsIdyeWKqzurgONT64Enjc+pkygwDLMSN2of3pqqTDMfRsladv2jl1qBBpX
AfOSVujgNE4QgscbGl8eUHU1Q6kQwEp94aqfrk/Ief4A3vwPDRpcZpSO5VMmGCcG
t/wjppZkCugNAEWI3t2m3Un5KgQlvdrT7/nSW8WGtcvG5y2VHveYrLQRRVIuLM1N
w5NKaqMhMB8wHQYDVR0OBBYEFKQizNqkQwKvcddRiFggh4+nbxvYMA0GCWCGSAFl
AwQDAgUAA0AAMD0CHQCsMcd773bfc5X3B4YMSU84rQoWgOeBqfe1PCVlAhxumOd0
rdNf82U2km0knr018vf7sQMdXWAamnmx
-----END CERTIFICATE-----
```

- We run the server by using the following command

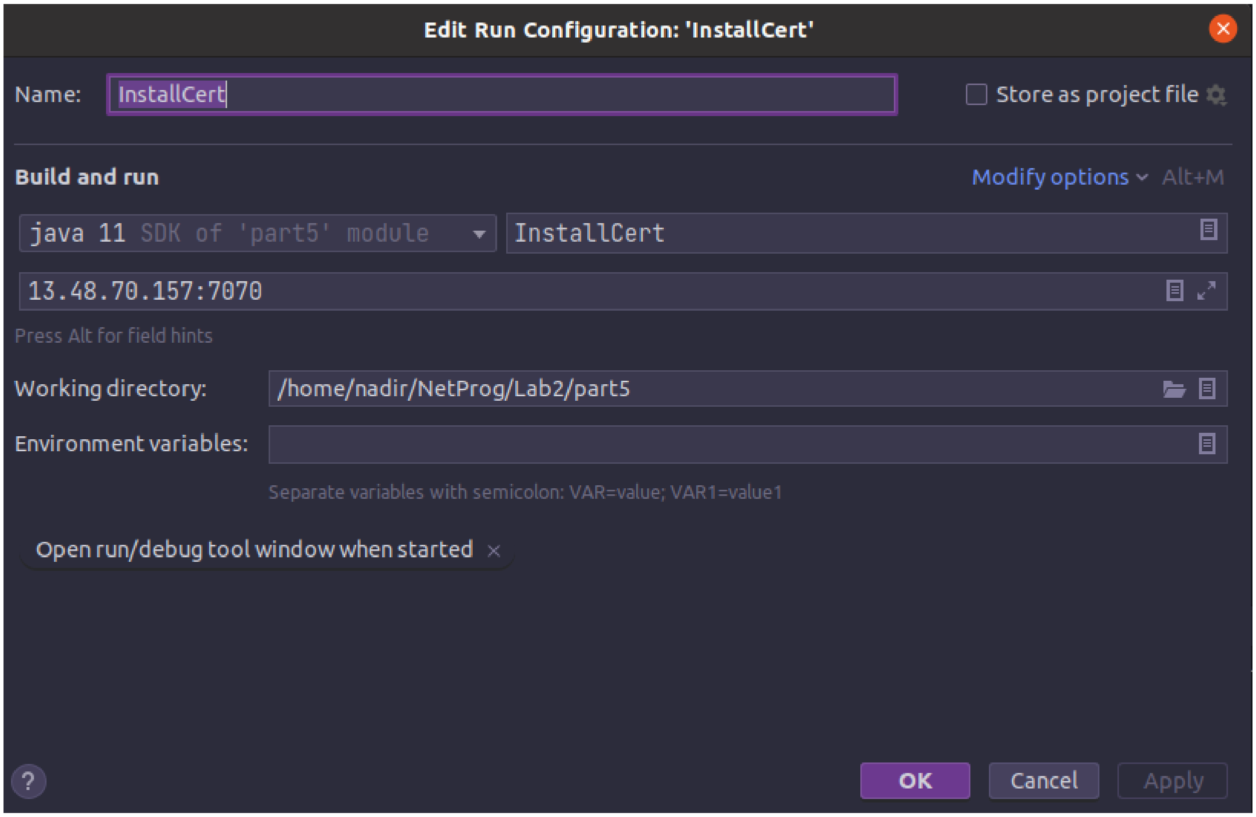
```
java -Djavax.net.ssl.keyStore=SSLStore -Djavax.net.ssl.keyStorePassword=nadir11
-Djdk.tls.server.protocols= TLSv1.2 LoginServer.java
```



```
ubuntu@ip-172-31-27-37:~/lab2$ java -Djavax.net.ssl.keyStor
e=SSLStore -Djavax.net.ssl.keyStorePassword=nadir11 -Djdk.t
ls.server.protocols=TLSv1.2 LoginServer.java
Waiting for connection...
```

2.2. Install Cert

- Once the certificate is generated at the server's side (in the cloud), we run the Install Cert class in the client's side to first start the SSL handshake and retrieve the certificate we generated in the previous step.



Edit Run Configuration: 'InstallCert'

Name: InstallCert

□ Store as project file

Build and run

java 11 SDK of 'part5' module

InstallCert

13.48.70.157:7070

Press Alt for field hints

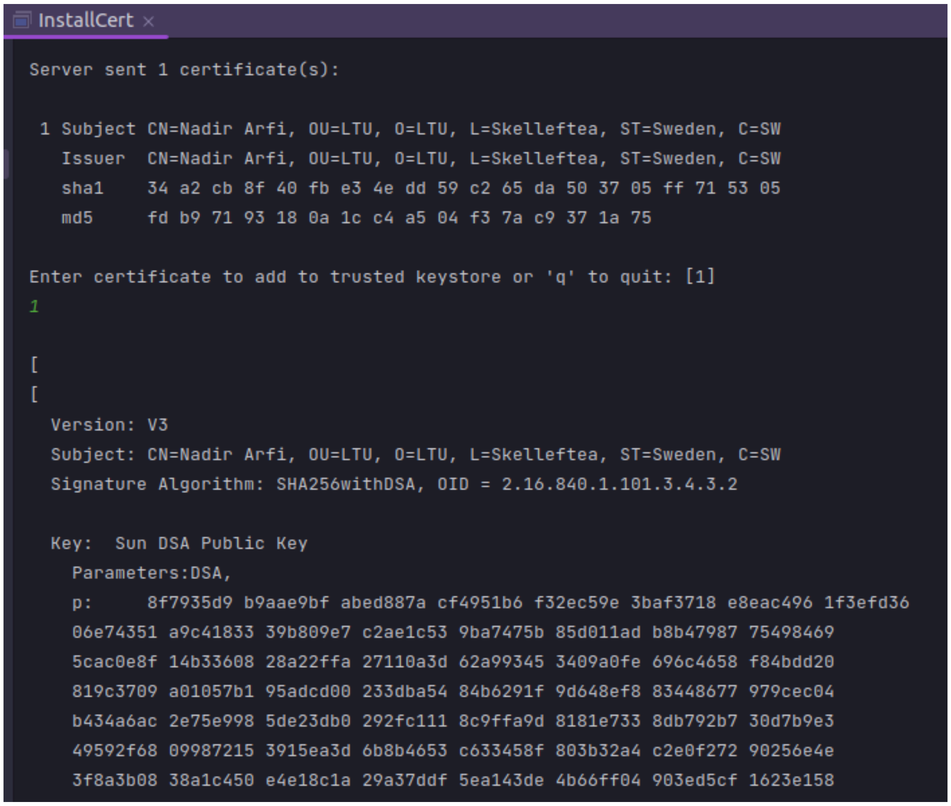
Working directory: /home/nadir/NetProg/Lab2/part5

Environment variables:

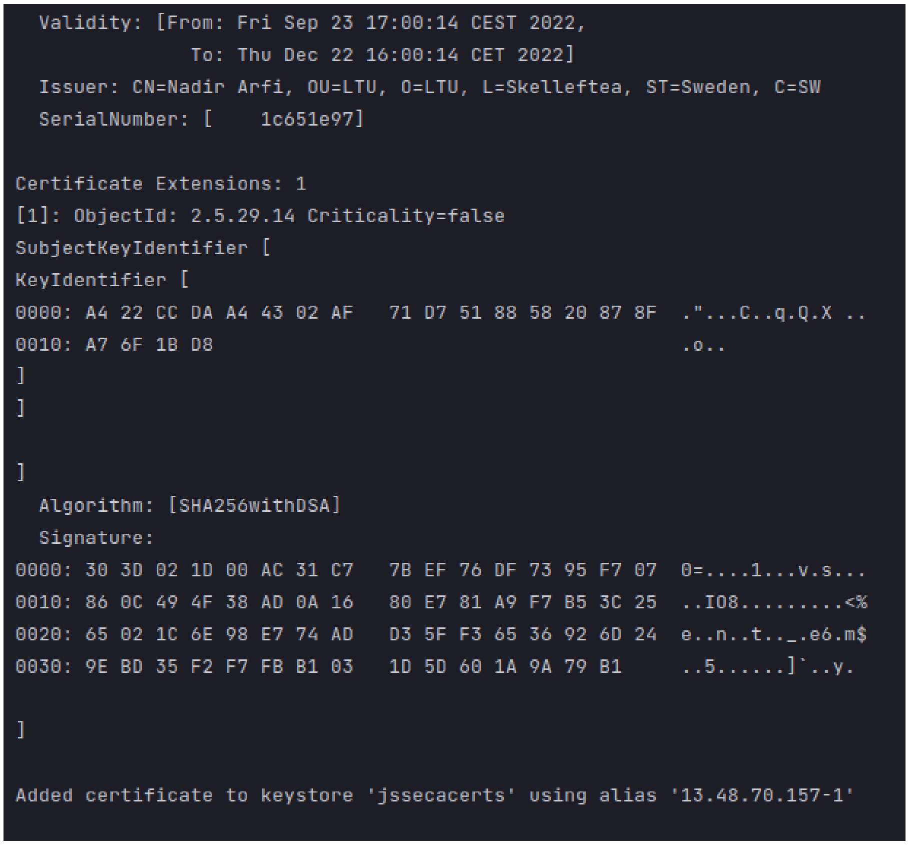
Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started ×

- As depicted in the figure below, the server sends 1 SSL certificate back to the client's side using an alias "13.48.70.157-1", representing the server's IP address. The certificate is added to the keyStore "jssecacerts".

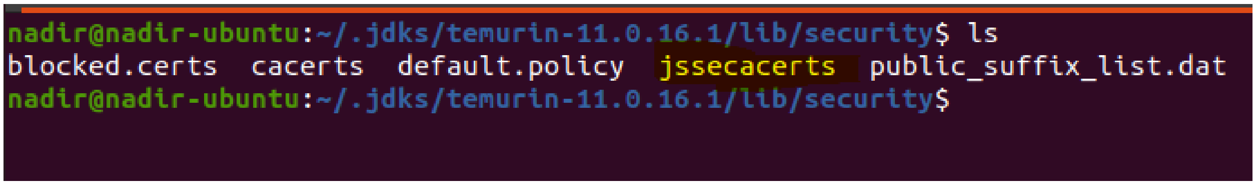


```
Server sent 1 certificate(s):

 1 Subject CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
   Issuer  CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
   sha1    34 a2 cb 8f 40 fb e3 4e dd 59 c2 65 da 50 37 05 ff 71 53 05
   md5     fd b9 71 93 18 0a 1c c4 a5 04 f3 7a c9 37 1a 75

Enter certificate to add to trusted keystore or 'q' to quit: [1]
1

[
[
  Version: V3
  Subject: CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
  Signature Algorithm: SHA256withDSA, OID = 2.16.840.1.101.3.4.3.2

  Key:  Sun DSA Public Key
    Parameters:DSA,
    p:     8f7935d9 b9aae9bf abed887a cf4951b6 f32ec59e 3baf3718 e8eac496 1f3efd36
    06e74351 a9c41833 39b809e7 c2ae1c53 9ba7475b 85d011ad b8b47987 75498469
    5cac0e8f 14b33608 28a22ffa 27110a3d 62a99345 3409a0fe 696c4658 f84bdd20
    819c3709 a01057b1 95adcd00 233dba54 84b6291f 9d648ef8 83448677 979cec04
    b434a6ac 2e75e998 5de23db0 292fc111 8c9ffa9d 8181e733 8db792b7 30d7b9e3
    49592f68 09987215 3915ea3d 6b8b4653 c633458f 803b32a4 c2e0f272 90256e4e
    3f8a3b08 38a1c450 e4e18c1a 29a37ddf 5ea143de 4b66ff04 903ed5cf 1623e158
```



```
  Validity: [From: Fri Sep 23 17:00:14 CEST 2022,
               To: Thu Dec 22 16:00:14 CET 2022]
  Issuer: CN=Nadir Arfi, OU=LTU, O=LTU, L=Skelleftea, ST=Sweden, C=SW
  SerialNumber: [    1c651e97]

Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: A4 22 CC DA A4 43 02 AF   71 D7 51 88 58 20 87 8F  ."...C..q.Q.X ..
0010: A7 6F 1B D8                                        .o..
]
]

]
  Algorithm: [SHA256withDSA]
  Signature:
0000: 30 3D 02 1D 00 AC 31 C7   7B EF 76 DF 73 95 F7 07  0=....1...v.s...
0010: 86 0C 49 4F 38 AD 0A 16   80 E7 81 A9 F7 B5 3C 25  ..IO8.........<%
0020: 65 02 1C 6E 98 E7 74 AD   D3 5F F3 65 36 92 6D 24  e..n..t.._.e6.m$
0030: 9E BD 35 F2 F7 FB B1 03   1D 5D 60 1A 9A 79 B1     ..5......]`..y.

]

Added certificate to keystore 'jssecacerts' using alias '13.48.70.157-1'
```

- However, this “jssecacerts” keystore is imported to the truststore of the client, which is located in $JAVA_HOME = /home/nadir/.jdks/temurin-11.0.16.1/lib/security

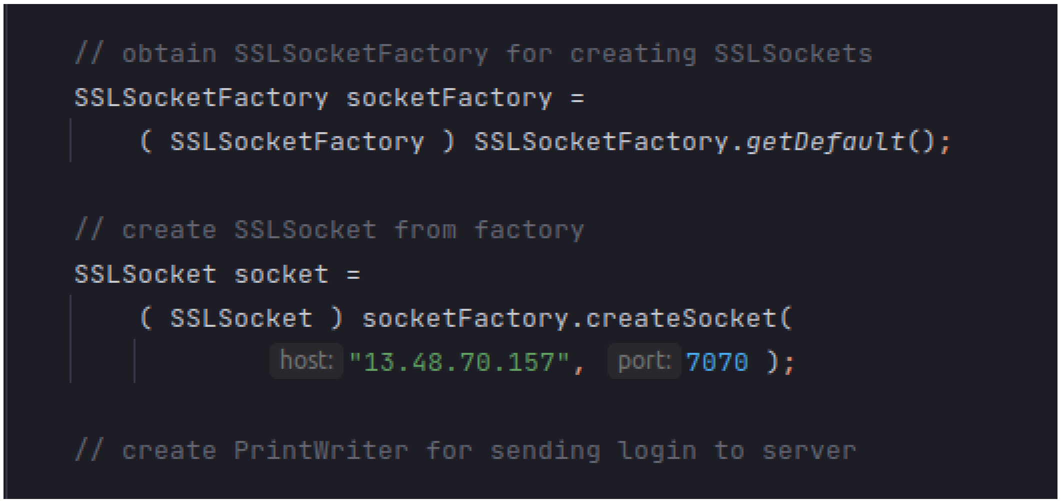


```
nadir@nadir-ubuntu:~/.jdks/temurin-11.0.16.1/lib/security$ ls
blocked.certs  cacerts  default.policy  jssecacerts  public_suffix_list.dat
nadir@nadir-ubuntu:~/.jdks/temurin-11.0.16.1/lib/security$
```

- This way, the client will be able to communicate with the server without necessarily having to use the keystore and the password every time.

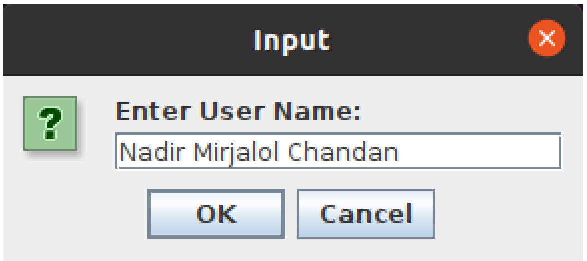
2.3. Login Client

- We make sure the client is connected to the correct public IP address of the server and its corresponding port



```
// obtain SSLSocketFactory for creating SSLSockets
SSLSocketFactory socketFactory =
    ( SSLSocketFactory ) SSLSocketFactory.getDefault();

// create SSLSocket from factory
SSLSocket socket =
    ( SSLSocket ) socketFactory.createSocket(
            host: "13.48.70.157",  port: 7070 );

// create PrintWriter for sending login to server
```

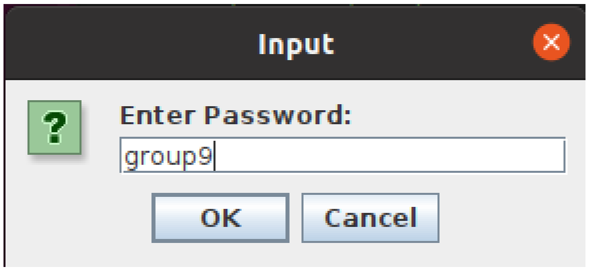
- We run the client, the client has to go through an authentication process where he must provide a username and a password



Input

Enter User Name:

Nadir Mirjalol Chandan

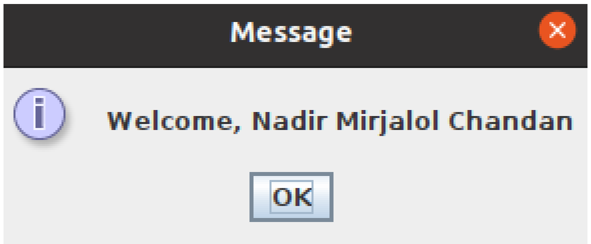


Input

Enter Password:

group9

- The credentials are transmitted to the server and if validated, the client is logged in.



Welcome, Nadir Mirjalol Chandan