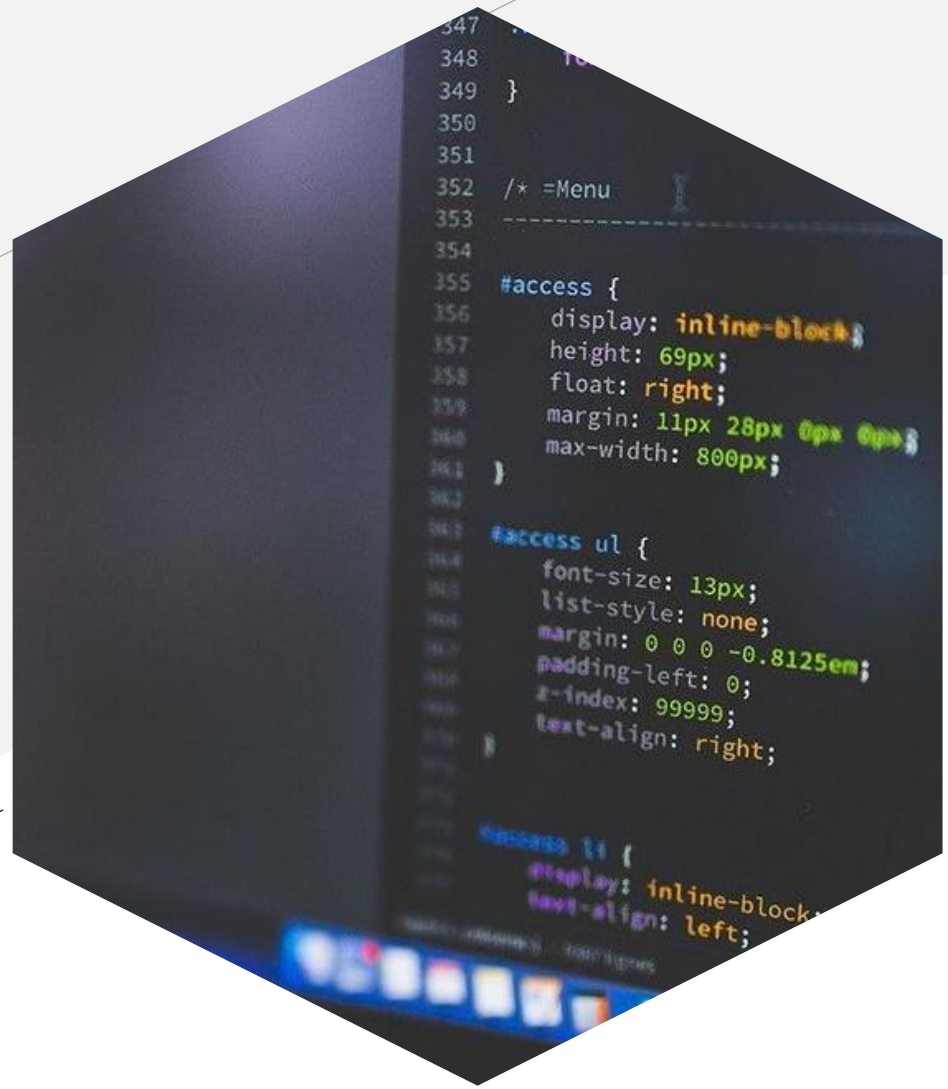


Network Programming HW 1

Batool D Shilleh

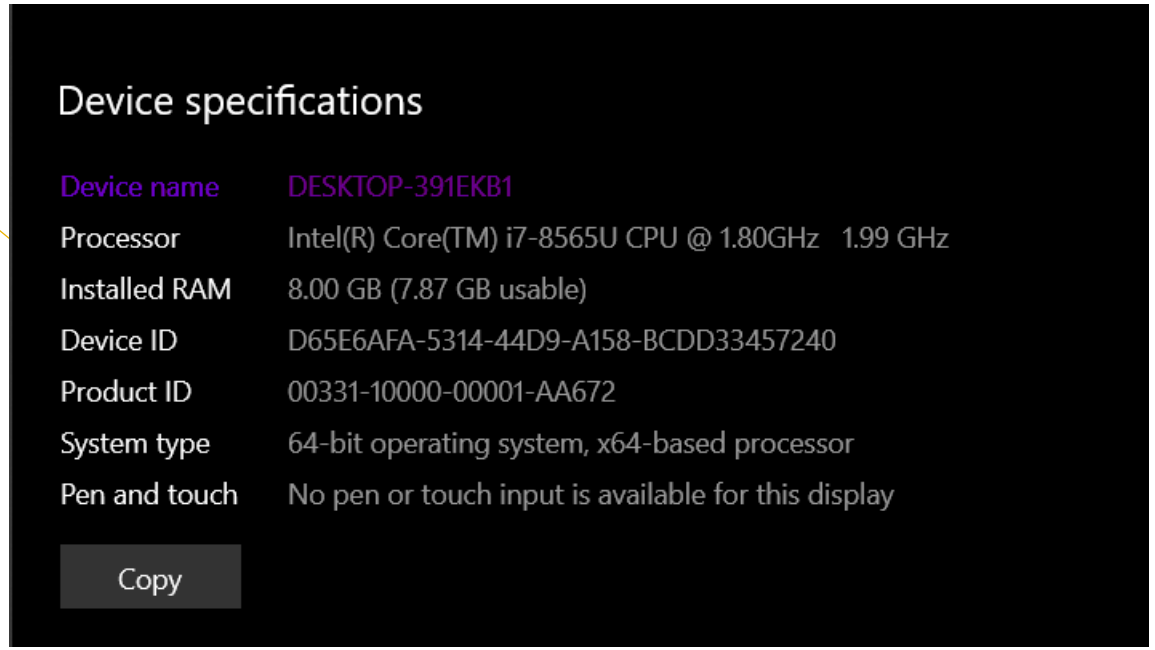
11923748

FR

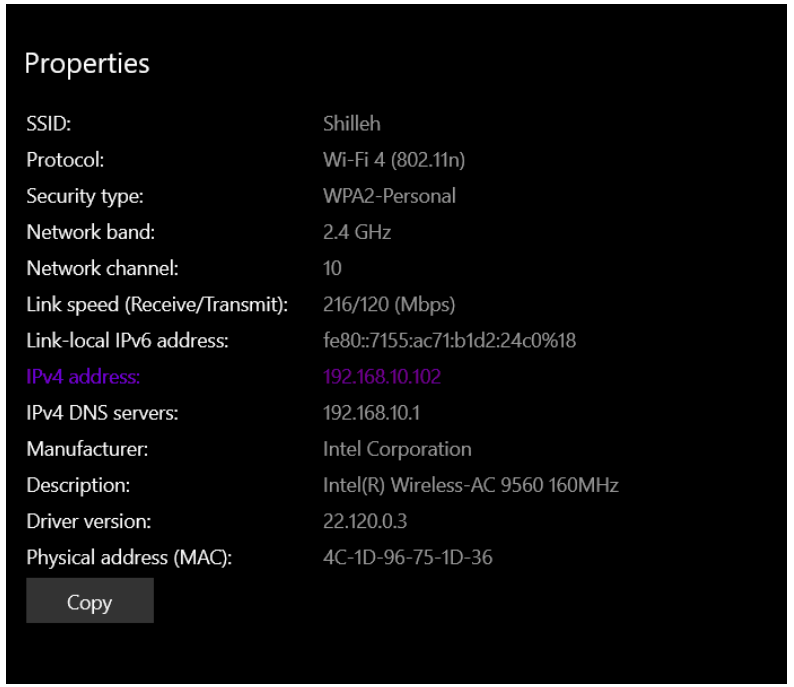
IP & HOST NAME

The images shows the name of the device & the ip

Host name



Host ip



fileReader class

This class is used in both codes, and it is the one that reads the file and returns the output

imports

This class is used in both codes, and it is the one that reads the file and returns the output

fileReader

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
```

Variations and Function

- This function takes the input from the user via the client class and returns an array of lists
- VehicleDetails :It stores one line of the file
- splittedLines: The value of the id is stored for later use
- VehicleIDsArray: The value of the id is stored
- VehicleArray: It stores the rest of the vehicle information
- Info :List that is returned
- Index :It is used to store the ID
- Found: A flag that determines whether the entry entered by the user is found or not

fileReader1

```
public class fileReader{
    public static ArrayList readFile(String incomingMessage) {
        String VehicleDetails;
        ArrayList<String> splittedLines = new ArrayList<String>();
        ArrayList<String>  VehicleIDsArray = new ArrayList<String>();
       ArrayList<String> VehicleArray = new ArrayList<String>();
       ArrayList<String> Info = new ArrayList<String>();
        int index=0;
        int found = 0;
```

readFile function

fileReader2

```
try {
File VehicleFile = new File("Vehicle.txt");
Scanner fileRead = new Scanner(VehicleFile);
while (fileRead.hasNextLine()) {

    VehicleDetails = fileRead.nextLine();
   String splitted []=VehicleDetails.split(" ");
    splittedLines.add(splitted[0]);


    VehicleIDsArray.add(splittedLines.get(index));

    index+=1;
    VehicleArray.add(VehicleDetails);




}
```

The file is opened and read, as long as I have a newline in the file:

- The first line of the file is stored in VehicleDetails
- This line is defragmented and every part of it is saved in splitted ,in this way [id,name,model,year,color]
- The identifier that is in the first position is taken into the splitted and stored in splittedLines
- The value that is in splittedLines is stored in VehicleIDsArray to preserve
- And each time we increase the index to move to the next location
- The rest of the information is stored in the VehicleArray

readFile function

- The value entered by the user is compared with all the values in the identifier list
  - If found, the values are added to the Info list that will be returned and raise the value of found to 1 and then return the list
  - If it is not found, it will return an error message after storing it in an array list

fileReader3

```
for(int i = 0; i <= 3; i++) {
    if(VehicleIDsArray.get(i).equals(incomingMessage)) {
        Info.add(VehicleArray.get(i));
        found = 1;
        return Info;
    }
}
if(found == 0) {
    String sen[] = {"Vehicle is not found !!"};

    for(int i = 0; i <= 4; i++) {
    Info.add(sen[i]);
    return Info;
    }
}
}
```

readFile function

Error message if the file is not found

fileReader4

```
    catch(FileNotFoundException e) {
            System.out.println("The Vehicle file cannot be found! Please make sure the
file already exists.");

        }
        return Info;

    }

}
```

TCPServer class

This class is used in both codes, and it is the one that reads the file and returns the output

TCPServer class

TCPServer

```
import java.io.*;
import java.net.*;
import java.util.ArrayList;
class TCPServer extends fileReader {

    public static void main(String argv[]) throws Exception

    {
        String clientSentence;

        ArrayList<String> Result=new ArrayList<String>();

        try (ServerSocket welcomeSocket = new ServerSocket(8808)) {

            while(true) {

            Socket connectionSocket = welcomeSocket.accept();

            BufferedReader inFromClient =

            new BufferedReader(new

            InputStreamReader(connectionSocket.getInputStream()));

            DataOutputStream outToClient =

            new DataOutputStream(connectionSocket.getOutputStream());
```

This code opens the socket through the added input number and defines a Arraylist to store the data to be displayed

TCPServer class

- In this part, the connection with the client begins and the input entered by the user is transferred to the server
- Then pass it to read the file
- Then it starts to list the list that was returned from the user
- Then the connection ends

TCPServer1

```
DataOutputStream outToClient =

            new DataOutputStream(connectionSocket.getOutputStream());

            clientSentence = inFromClient.readLine();

            Result =readFile(clientSentence);

            for(int i=0; i<=4;i++) {
                outToClient.writeBytes(Result.get(i));

            outToClient.close();

            inFromClient.close();

            connectionSocket.close();
            }

            }
        }
    }
}
```

TCPClient class

This class is used in both codes, and it is the one that reads the file and returns the output

TCPClient class

TCPClient

```
package aa;
import java.io.*;
import java.net.*;

    class TCPClient  {

        public static void main(String argv[])
        {

            System.out.println("Hi there! You can now start sending requests to
      the server.");

            try {

                while (true) {

            String sentence;

            String modifiedSentence;

            BufferedReader inFromUser =

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

            Socket clientSocket = new Socket("192.168.10.102",8808);
```

- print sentence
- It starts by establishing the connection and taking the client name and the entrance number

TCPClient class

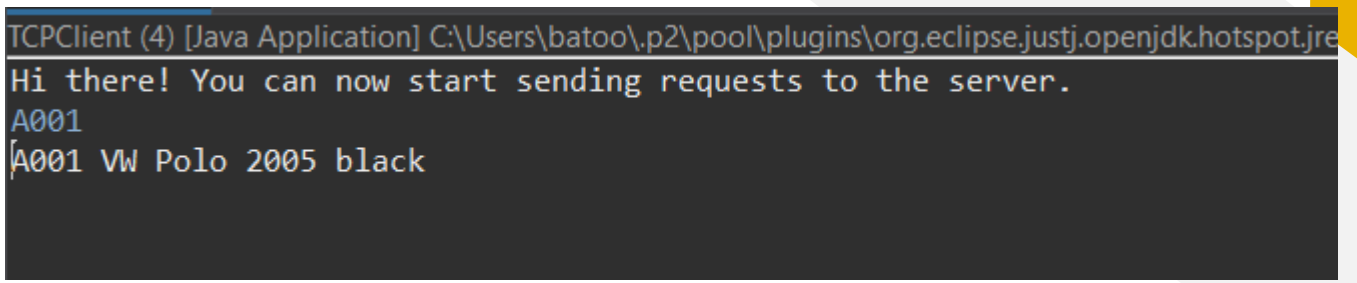
- Take information from the console
- Pass it to the server

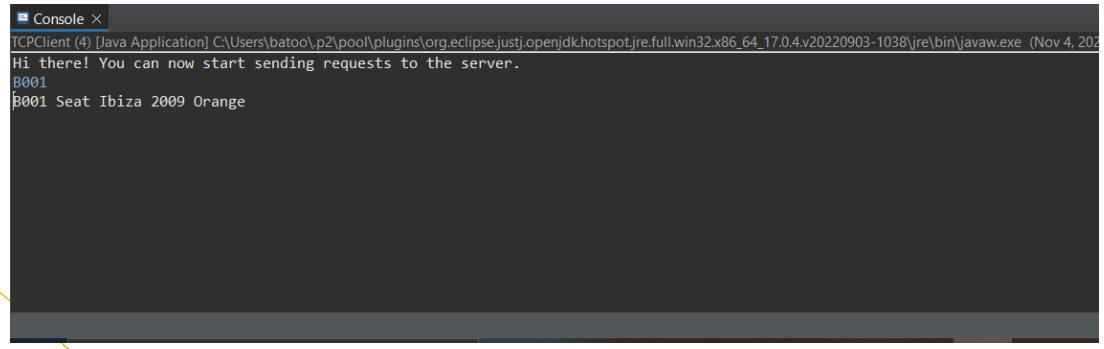
TCPClient

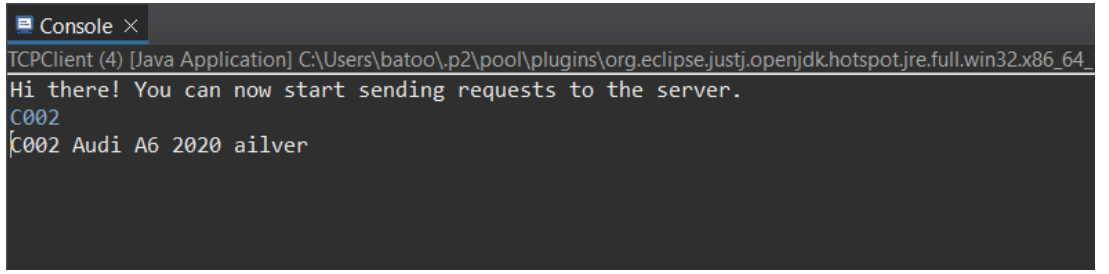
```
DataOutputStream outToServer =

            new DataOutputStream(clientSocket.getOutputStream());

            BufferedReader inFromServer =

            new BufferedReader(new

            InputStreamReader(clientSocket.getInputStream()));

            sentence = inFromUser.readLine();

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

            modifiedSentence = inFromServer.readLine();

            System.out.println(modifiedSentence);

            clientSocket.close();

                }

            }

            catch (Exception e) {

                e.printStackTrace();
            }
        }
    }
```

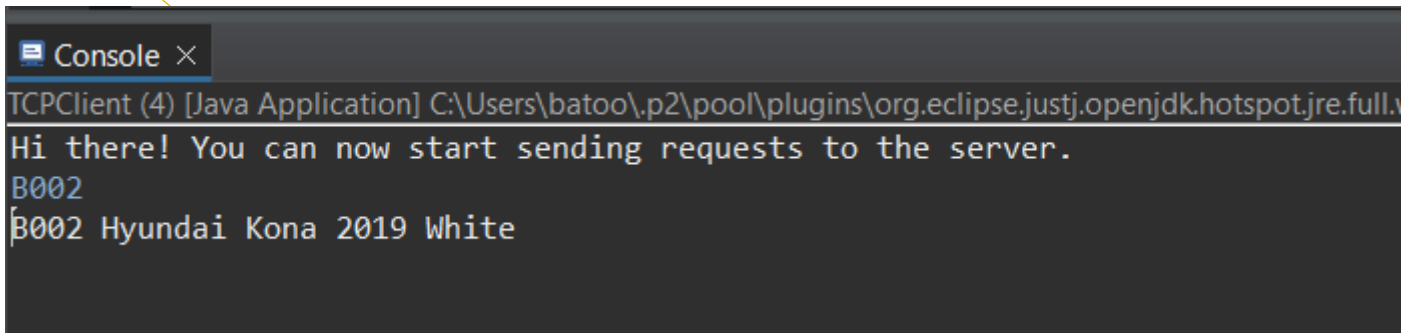
OUT PUT TCP

FR



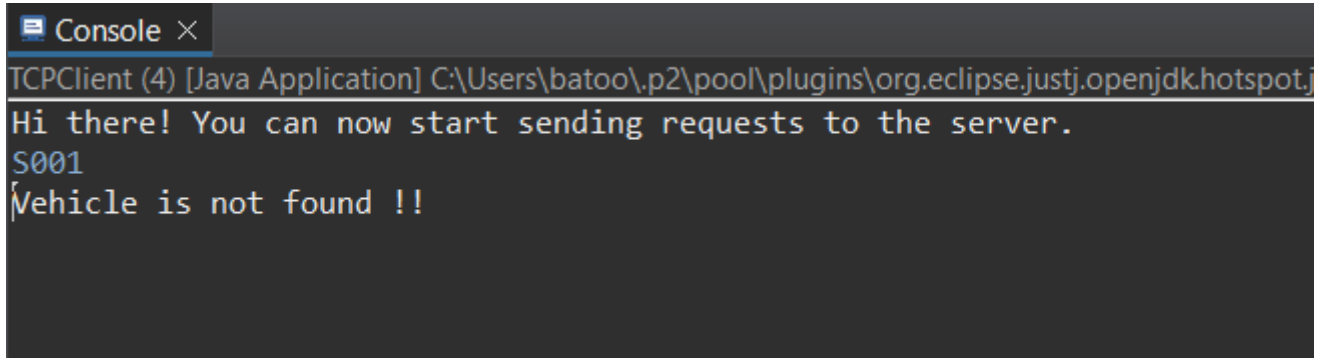




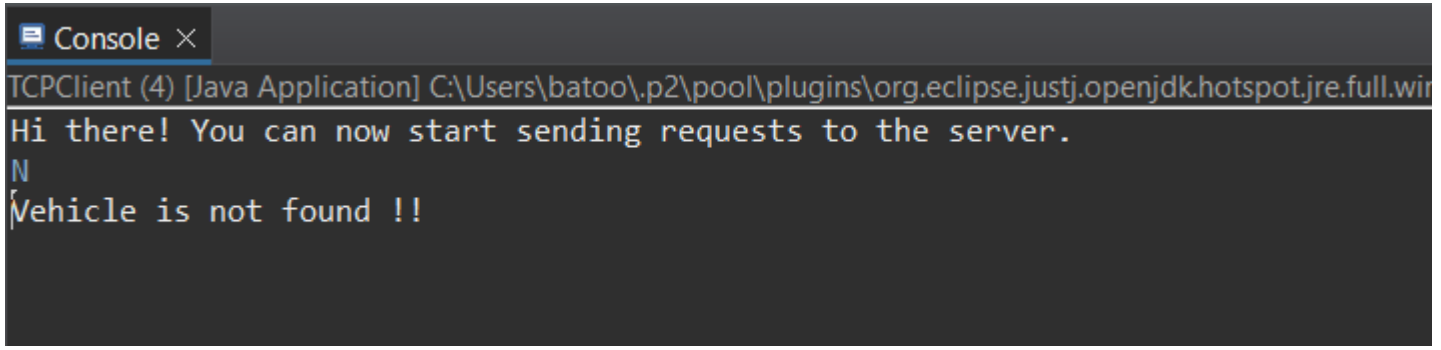


It is in the file

OUT PUT TCP



Not found in the file



UDPClient class

This class is used in both codes, and it is the one that reads the file and returns the output

UDPClient class

- building datagram
- And pass the username to start the connection
- And read the information from the clients

UDPClient

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String args[])
    {

        System.out.println("Hi there! You can now start sending datagram packets to the
server.");

        try {

            while (true) {

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

                DatagramSocket clientSocket = new DatagramSocket();

                InetAddress IPAddress = InetAddress.getByName("DESKTOP-391EKB1");

                byte[] sendData = new byte[1024];

                byte[] receiveData = new byte[1024];

                String sentence = inFromUser.readLine();

                sendData = sentence.getBytes();
```

UDPClient class

UDPClient 1

```
DatagramPacket sendPacket =
                new DatagramPacket(sendData, sendData.length, IPAddress, 7775);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                new String(receivePacket.getData());

                System.out.println(modifiedSentence);

                clientSocket.close();

                    }

                }

                catch (Exception e) {

                e.printStackTrace();
        }
    }
}
```

- Passing information to constructer the datagram
- The connection with the server started

UDPServer class

This class is used in both codes, and it is the one that reads the file and returns the output

UDPServer

- Here we will not create a socket, but we will build a datagram
- We will pass the input number and client name
- We create an array to store the data

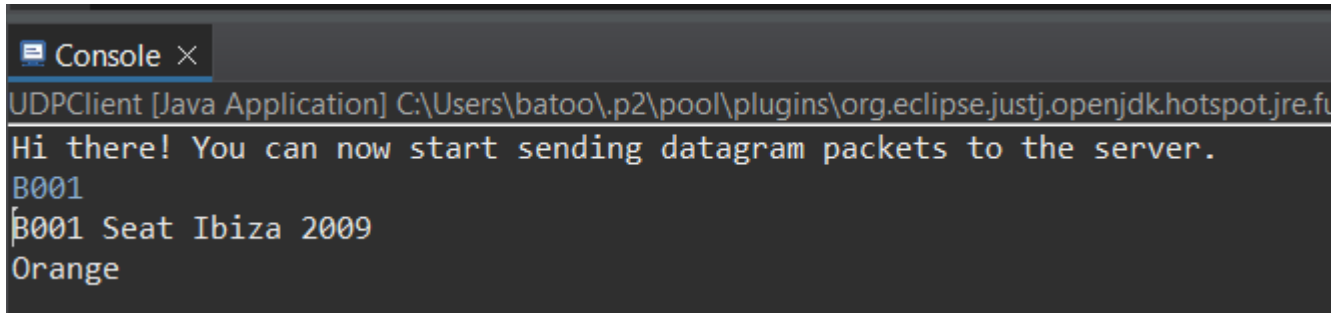
UDPServer

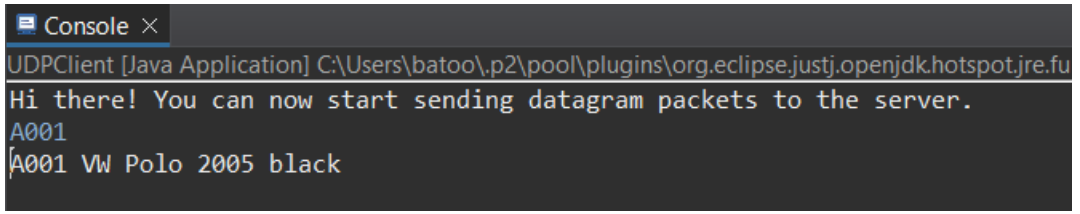
```
package aa;

import java.io.*;
import java.net.*;
import java.util.ArrayList;
class UDPServer extends fileReader {

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

    {

    try (DatagramSocket serverSocket = new DatagramSocket(7775)) {

            byte[] receiveData = new byte[1024];

            byte[] sendData;

            while(true)

            {

                try {

                    DatagramPacket datagramPacket =
                            new DatagramPacket(receiveData, receiveData.length);
```

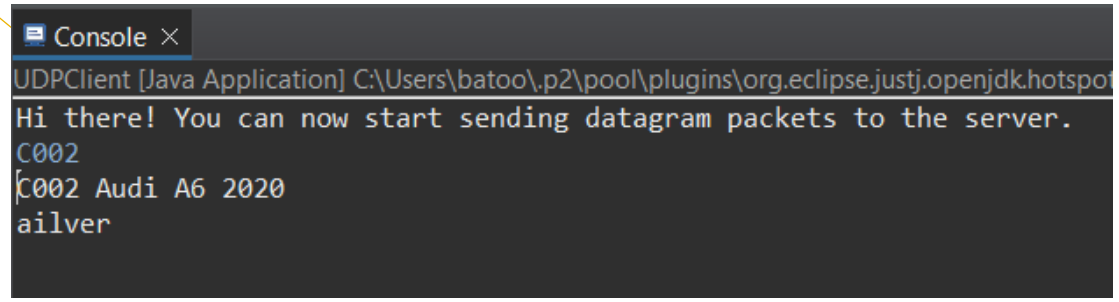
UDPServer

- Passing the input from the user to read the file
- Return the output from reading and printing the file

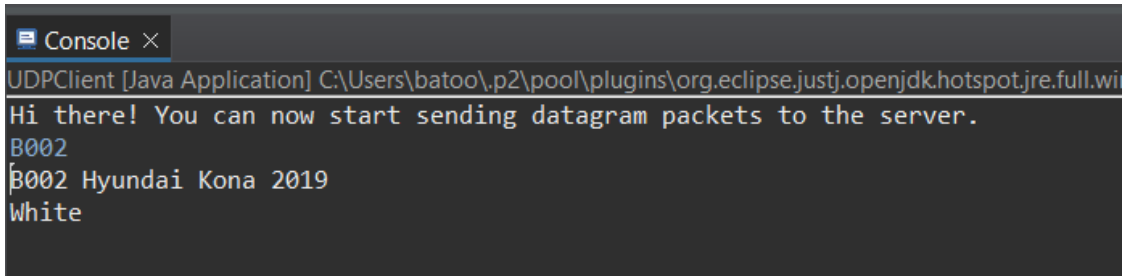
OUT PUT UDP



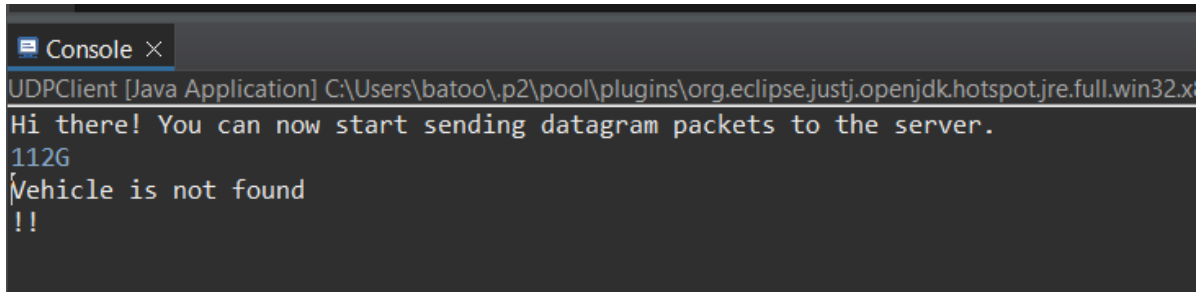




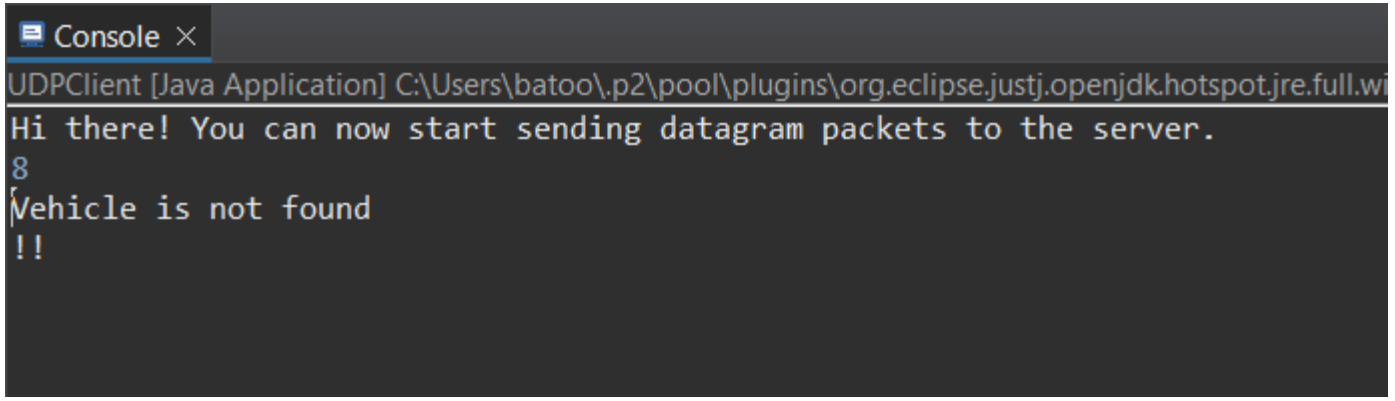
It is in the file



OUT PUT UDP



Not found in the file



THE FILE

The file in which the information is stored

Vehicle.txt

Vehicle.txt

```
B001 Seat Ibiza 2009 Orange
B002 Hyundai Kona 2019 White
A001 VW Polo 2005 black
C002 Audi A6 2020 ailver
```

Clients info

The code was run on two devices, and this is the client's information

Clients info

```
Pinging 127.0.0.1 with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```