Δίκτυα Υπολογιστών II

Ιωαννης-Παναγιωτης Μπουντουριδης

ΑΕΜ: 8872

source

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Scanner;
import java.net.*;
import java.io.*;
import java.util.ArrayList;


public class userApplication {
    public static void main(String[] param) throws LineUnavailableException {
        // session id
        int serverPort =    38002 ; // MUST BE FILLED
        int clientPort =     48002 ; // MUST BE FILLED
        int echo_code_delay =5525 ; // MUST BE FILLED
        int image_code = 9624; // MUST BE FILLED
        int sound_code = 7230; // MUST BE FILLED
        int vehicle_code = 3489; // MUST BE FILLED
        // initialize scanner
        Scanner scanner = new Scanner(System.in);
        //time to choose
        int userChoice = 0;
        // printing choices
        System.out.print("\n1. Create Datagrams G1,G2");
        System.out.print("\n2. Create Datagrams G3,G4");
        System.out.print("\n3. Create Image E1");
        System.out.print("\n4. Create Image E2");
        System.out.print("\n5. Create Temperatures");
        System.out.print("\n6. Create DPCM request song");
        System.out.print("\n7. Create DPCM request freq");
        System.out.print("\n8. Create AQDPCM request song");
        System.out.print("\n9. Create AQDPCM request freq");
        System.out.print("\n10. Vehicle..");
        // get user choice
        userChoice = Integer.parseInt(scanner.nextLine());
        // apply
        if (userChoice == 1 || userChoice == 2) {
            try {
                echo_choice(echo_code_delay, userChoice, serverPort, clientPort);
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else if (userChoice == 3 || userChoice == 4) {
            try {
                image_choice(image_code, userChoice, serverPort, clientPort);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        else if (userChoice == 5){
            try {
                temperatures_choice(echo_code_delay, serverPort, clientPort);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        else if(userChoice == 6 || userChoice == 7){
            try{
                DPCM_choice(sound_code,userChoice,serverPort,clientPort);
            } catch (SocketException e) {
                e.printStackTrace();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        else if(userChoice == 8 || userChoice == 9){
            try{
                AQDPCM_choice(sound_code,userChoice,serverPort,clientPort);
            } catch (SocketException e) {
                e.printStackTrace();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        else if (userChoice == 10){
            try{
                vehicle_choice(vehicle_code,serverPort,clientPort,"0C");
            } catch (SocketException e) {
                e.printStackTrace();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
    // user cases 1 and 2
    public static void echo_choice(int echo_code, int echo_case, int server_listening_port, int client_listening_port) throws SocketException,
        // used to print echo packet messages
        String echo_message = "";
        // gives the correct name to output files for both cases
        String chosen_mode = "";
        // echo packet code from the session
        String code = "";
        switch (echo_case){
            case 1:
                chosen_mode = "case_delay";
                code = "E" + Integer.toString(echo_code) + "\r";
                break;
            case 2:
                chosen_mode = "case_without_delay";
                code = "E0000\r";
                break;
        }
        //init InetAddress
        byte[] hostIP = {(byte) 155, (byte) 207, 18, (byte) 208};
        InetAddress hostAddress = InetAddress.getByAddress(hostIP);
        // getArray of bytes
        byte[] code_byte = code.getBytes();
        // init DatagramSocket
        DatagramSocket sendSocket = new DatagramSocket();
        DatagramPacket sendPacket = new DatagramPacket(code_byte, code_byte.length, hostAddress, server_listening_port);
        DatagramSocket recieveSocket = new DatagramSocket(client_listening_port);
        recieveSocket.setSoTimeout(3600);
        // init arrayOf bytes
        byte[] recieve_byte = new byte[2048];
        DatagramPacket recievePacket = new DatagramPacket(recieve_byte, recieve_byte.length);
        // time of packets will be saved here
        ArrayList<Double> arrayList_timePacket = new ArrayList<Double>();
        // we will count all packets recieved
```

```
int packet_counter = 0;
// time variables
double timeElapsed = 0;
double averageTime = 0;
double beginSession = 0;
double endSession = 0;
double beginTime = 0;
double endTime = 0;
// get current time
beginSession = System.nanoTime();
// set time for recieving packets
int five_minutes_session = 5 * 60 * 1000;
// results of session will be saved here
ArrayList<String> arrayList_output = new ArrayList<String>();
// begin loop
while (endSession < five_minutes_session) {
    //sending one packet
    sendSocket.send(sendPacket);
    // update time
    beginTime = System.nanoTime();
    // increase counter
    packet_counter++;
    for (; ; ) {
        try {
            // recieve packet
            recieveSocket.receive(recievePacket);
            // update end time
            endTime = (System.nanoTime() - beginTime) / 1000000;
            // print message
            echo_message = new String(recieve_byte, 0, recievePacket.getLength());
            System.out.println(echo_message);
            // print time response
            System.out.print("" + endTime + "\n");
            break;
        } catch (Exception x) {
            System.out.println(x);
            break;
        }
    }
    // update time elapsed
    timeElapsed += endTime;
    // add endTime
    arrayList_timePacket.add(endTime);
    // update ArrayList for output file
    arrayList_output.add("" + endTime);
    // update endSession
    endSession = (System.nanoTime() - beginSession) / 1000000;
}
// some handy statistics for our report
ArrayList<String> arrayList_stats = new ArrayList<String>();
// average time
averageTime = timeElapsed / packet_counter;
arrayList_stats.add("Session time :" + (timeElapsed / 60) / 1000 + " minutes\n");
arrayList_stats.add("Total time : " + (endSession / 60) / 1000 + " minutes\n");
arrayList_stats.add("Total number of packets : " + String.valueOf((double) packet_counter));
arrayList_stats.add("Total average time : " + String.valueOf(averageTime));
double sum = 0;
float counter = 0;
ArrayList<Float> counters = new ArrayList<Float>();
for (int i = 0; i < arrayList_timePacket.size(); i++) {
    int j = i;
    while ((sum < 8 * 1000) && (j < arrayList_timePacket.size())) {
        sum += arrayList_timePacket.get(j);
        counter++;
        j++;
    }
    counter = counter / 8;
    counters.add(counter);
    counter = 0;
    sum = 0;
}
// create output file
BufferedWriter bufferedWriter = null;
try {
    String fileDestination = "Echo_" + echo_code + "_" + chosen_mode + ".txt";
    File file = new File(fileDestination);
    bufferedWriter = new BufferedWriter(new FileWriter(fileDestination, false));
    if (!file.exists()) {
        file.createNewFile();
    }
    for (int i = 0; i < arrayList_output.size(); i++) {

        bufferedWriter.write(String.valueOf(arrayList_output.get(i)));
        bufferedWriter.newLine();
    }
    bufferedWriter.newLine();
} catch (IOException ioe) {
    ioe.printStackTrace();
} finally {
    try {
        if (bufferedWriter != null) bufferedWriter.close();
    } catch (Exception ex) {
        System.out.println("Error in closing the BufferedWriter" + ex);
    }
}

bufferedWriter = null;

try {
    String fileDestination = "Echo_Statistics_" + echo_code + "_" + chosen_mode + ".txt";
    File file = new File(fileDestination);
    bufferedWriter = new BufferedWriter(new FileWriter(fileDestination, false));
    if (!file.exists()) {
        file.createNewFile();
    }
    for (int i = 0; i < arrayList_stats.size(); i++) {
        bufferedWriter.write(String.valueOf(arrayList_stats.get(i)));
        bufferedWriter.newLine();
    }
    bufferedWriter.newLine();
} catch (IOException ioe) {
    ioe.printStackTrace();
} finally {
    try {
        if (bufferedWriter != null) bufferedWriter.close();
    } catch (Exception ex) {
        //TODO
    }
}
bufferedWriter = null;
try {
    String finalDestination = "Echo_R_" + echo_code + "_" + chosen_mode + ".txt";
    File file = new File(finalDestination);
    bufferedWriter = new BufferedWriter(new FileWriter(finalDestination, false));
    if (!file.exists()) {
        file.createNewFile();
    }
    for (int i = 0; i < counters.size(); i++) {
        bufferedWriter.write(String.valueOf(counters.get(i)));
        bufferedWriter.newLine();
```

```
            }
            bufferedWriter.newLine();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        } finally {
            try {
                if (bufferedWriter != null) bufferedWriter.close();
            } catch (Exception ex) {
                //TODO
            }
        }
        recieveSocket.close();
        sendSocket.close();
    }
    // User cases 3 and 4
    public static void image_choice(int img_code, int case_image, int server_listening_port, int client_listening_port) throws SocketException,
        // update image code based on every case
        String image_code = "";
        // update title for file
        String title_case = "";
        switch (case_image){
            case 3:
                title_case = "CAMERA_1";
                image_code = "M" + Integer.toString(img_code) + "\r";
                break;
            case 4:
                title_case = "CAMERA_2";
                image_code = "M" + Integer.toString(img_code) + " " + "CAM=PTZ" + "\r";
                break;
        }
        // initialize InetAddress
        byte[] hostIP = {(byte) 155, (byte) 207, 18, (byte) 208};
        InetAddress hostAddress = InetAddress.getByAddress(hostIP);
        // array of bytes
        byte[] code_array = image_code.getBytes();
        // initialize DatagramSocket
        DatagramSocket sendSocket = new DatagramSocket();
        DatagramPacket sendPacket = new DatagramPacket(code_array, code_array.length, hostAddress, server_listening_port);
        DatagramSocket recieveSocket = new DatagramSocket(client_listening_port);
        recieveSocket.setSoTimeout(3600);
        // array of bytes
        byte[] byte_recieve_array = new byte[2048];
        DatagramPacket recievePacket = new DatagramPacket(byte_recieve_array, byte_recieve_array.length);
        sendSocket.send(sendPacket);
        // set Timeout
        recieveSocket.setSoTimeout(3200);
        // output file name
        String outputName = ("image" + img_code + title_case + ".jpeg");
        // initialize File-Output-Stream
        FileOutputStream fOS = new FileOutputStream(outputName);
        for (; ; ) {
            try {
                // receiving packets
                recieveSocket.receive(recievePacket);
                if (byte_recieve_array == null) break;
                for (int i = 0; i <= 127; i++) {
                    fOS.write(byte_recieve_array[i]);
                }
            } catch (IOException ex) {
                System.out.println(ex);
                break;
            }
        }
        fOS.close();
        recieveSocket.close();
        sendSocket.close();
    }
    // User case 5
    public static void temperatures_choice(int echoCode, int server_listening_port, int client_listening_port) throws SocketException, IOExcept
        String packetInfo = "";
        String code = "E" + Integer.toString(echoCode) + "\r";
        byte[] hostIP = {(byte) 155, (byte) 207, 18, (byte) 208};
        InetAddress hostAddress = InetAddress.getByAddress(hostIP);
        // getArray of bytes
        byte[] code_byte = code.getBytes();
        // init DatagramSocket
        DatagramSocket sendSocket = new DatagramSocket();
        DatagramPacket sendPacket = new DatagramPacket(code_byte, code_byte.length, hostAddress, server_listening_port);
        DatagramSocket recieveSocket = new DatagramSocket(client_listening_port);
        recieveSocket.setSoTimeout(3600);
        // init arrayOf bytes
        byte[] recieve_byte = new byte[2048];
        DatagramPacket recievePacket = new DatagramPacket(recieve_byte, recieve_byte.length);
        // time of packets will be saved here
        ArrayList<Double> arrayList_timePacket = new ArrayList<Double>();
        // we will count all packets recieved
        int packet_counter = 0;
        // time variables
        double timeElapsed = 0;
        double averageTime = 0;
        double beginSession = 0;
        double endSession = 0;
        double beginTime = 0;
        double endTime = 0;
        // get current time
        beginSession = System.nanoTime();
        int numberOfPackets = 0;
        String message = "";
        beginSession = System.nanoTime();
        for (int i = 0; i <= 9; i++) {
            packetInfo = "E" + Integer.toString(echoCode) + "T0" + i + "\r";
            code_byte = packetInfo.getBytes();
            sendPacket = new DatagramPacket(code_byte, code_byte.length, hostAddress, server_listening_port);
            sendSocket.send(sendPacket);
            numberOfPackets++;
            beginTime = System.nanoTime();
            for (; ; ) {
                try {
                    recieveSocket.receive(recievePacket);
                    endTime = (System.nanoTime() - beginTime) / 1000000;
                    message = new String(recieve_byte, 0, recievePacket.getLength());
                    System.out.println(message);
                    System.out.print("" + endTime + "\n");
                    break;
                } catch (Exception x) {
                    System.out.println(x);
                    break;
                }
            }
            timeElapsed += endTime;
            endSession = (System.nanoTime() - beginSession) / 1000000;
        }
    }
    //User cases 6 and 7
    public static void DPCM_choice(int audioCode,int mode,int serverPort,int clientPort) throws SocketException,IOException,UnknownHostExceptio
        // package request case
        String pRequest = "";
        // name request title
        String nameCase="";
        // setup attributes
```

```
    switch (mode){
        case 6:
            nameCase="SONG";
            pRequest = "A" + Integer.toString(audioCode) + "F999";
            break;
        case 7:
            nameCase="FREQ";
            pRequest = "A" + Integer.toString(audioCode) + "T999";
            break;
    }
    // declare number of packets
    int totalPackets = 999;
    // inet addresss initialize()
    byte[] hostIP = { (byte)155,(byte)207,18,(byte)208 };
    InetAddress hostAddress = InetAddress.getByAddress(hostIP);
    // package code array of bytes
    byte[] code_array = pRequest.getBytes();
    // Datagram initialize()
    DatagramSocket sendSocket = new DatagramSocket();
    DatagramPacket sendPacket = new DatagramPacket(code_array,code_array.length, hostAddress,serverPort);
    DatagramSocket recieveSocket = new DatagramSocket(clientPort);
    recieveSocket.setSoTimeout(3600);
    // initialize array of bytes
    byte[] bytes_recieved = new byte[128];
    DatagramPacket recievePacket = new DatagramPacket(bytes_recieved,bytes_recieved.length);
    sendSocket.send(sendPacket);
    // initialize array of bytes
    byte[] song = new byte[256*totalPackets];
    // declare nibble variables
    int nbL,nbH;
    // reformed nibbles
    int reFnb1,reFnb2;
    // defines possition of array
    int pos = 0;
    int x1 = 0,x2 = 0;
    ArrayList<Integer> sb = new ArrayList<Integer>();
    ArrayList<Integer> smpl = new ArrayList<Integer>();
    for(int i = 1;i < totalPackets;i++){
        try{
            recieveSocket.receive(recievePacket);
            for (int j = 0;j <= 127;j++){
                //15 binary representation is 00001111
                nbL = bytes_recieved[j] & 15;
                //240 binary representation is 11110000
                nbH = (bytes_recieved[j] & 240)>>4;
                // reform nibble
                reFnb1 = nbL-8;
                // store
                sb.add(reFnb1);
                reFnb1 = reFnb1*5;
                // reform nibble
                reFnb2 = nbH-8;
                // store
                sb.add(reFnb2);
                reFnb2 = reFnb2*5;
                x1 = x2 + reFnb1;
                smpl.add(x1);
                x2 = x1 + reFnb2;
                smpl.add(x2);
                song[pos] = (byte)x1;
                // inc pos
                pos++;
                song[pos] = (byte)x2;
                // inc pos
                pos++;
            }
        }catch (Exception exception){
           //fatal exception
        }
        if((i%250)==0){
            System.out.println((1000-i)+" left");
        }
    }
    if(mode==6){
        AudioFormat pcm = new AudioFormat(8000,8,1,true,false);
        SourceDataLine startPlaying = AudioSystem.getSourceDataLine(pcm);
        startPlaying.open(pcm,32000);
        startPlaying.start();
        startPlaying.write(song,0,256*totalPackets);
        startPlaying.stop();
        startPlaying.close();
        System.out.println("Song started.");
    }
    // export dcpm sub list
    saveSubList(audioCode,nameCase,sb);
    // export dcpm sample list
    saveSampleList(audioCode,nameCase,smpl);
    // disconnecting
    recieveSocket.close();
    sendSocket.close();
}
//exports data
static void saveSubList(int audioCode,String nameCase,ArrayList sb){
    BufferedWriter bufferedWriter = null;
    try{
        File f = new File("DPCM_SUB_CODE_"+audioCode+"_"+nameCase+".txt");
        if(!f.exists()){
            f.createNewFile();
        }
        // append false
        FileWriter fileWriter = new FileWriter(f,false);
        bufferedWriter = new BufferedWriter(fileWriter);
        for(int i = 0 ; i < sb.size() ; i += 2){
            bufferedWriter.write("" + sb.get(i) + " " + sb.get(i+1));
            bufferedWriter.newLine();
        }
    }catch(IOException ioe){
        ioe.printStackTrace();
    }finally{
        try{
            if(bufferedWriter != null) bufferedWriter.close();
        }catch(Exception ex){
        }
    }
}
// exports data
static void saveSampleList(int audioCode,String nameCase,ArrayList smpl){
    BufferedWriter bufferedWriter = null;
    try{
        File file = new File("DPCM_SAMPLE_CODE_"+audioCode+"_"+nameCase+".txt");
        if(!file.exists()){
            file.createNewFile();
        }
        FileWriter fileWriter = new FileWriter(file,false);
        bufferedWriter = new BufferedWriter(fileWriter);
        for(int i = 0 ; i < smpl.size() ; i += 2){
            bufferedWriter.write("" + smpl.get(i) + " " + smpl.get(i+1));
            bufferedWriter.newLine();
        }
    }catch(IOException ioe){
```

```
        ioe.printStackTrace();
    }finally{
        try{
            if(bufferedWriter != null) bufferedWriter.close();
        }catch(Exception ex){
        }
    }
}
// User cases 8 and 9
public static void AQDPCM_choice(int audioCode,int mode,int serverPort,int clientPort) throws SocketException,IOException,UnknownHostExcept
    // package request case
    String pRequest = "";
    // package request title name
    String caseName="";
    // setup attributes
    switch (mode){
        case 8:
            caseName="SONG";
            pRequest = "A" + Integer.toString(audioCode) + "AQF999";
            break;
        case 9:
            caseName="FREQ";
            pRequest = "A" + Integer.toString(audioCode) + "AQT999";
            break;
    }
    // inet address
    byte[] hostIP = { (byte)155,(byte)207,18,(byte)208 };
    InetAddress hostAddress = InetAddress.getByAddress(hostIP);
    // packet code bytes array
    byte[] code_array = pRequest.getBytes();
    // datagram send initialize()
    DatagramSocket sendSocket = new DatagramSocket();
    DatagramPacket sendPacket = new DatagramPacket(code_array,code_array.length, hostAddress,serverPort);
    // datagram recieve initialize()
    DatagramSocket recieveSocket = new DatagramSocket(clientPort);
    byte[] recieveBuffer = new byte[132];
    DatagramPacket recievePacket = new DatagramPacket(recieveBuffer,recieveBuffer.length);
    recieveSocket.setSoTimeout(5000);
    sendSocket.send(sendPacket);
    // array of bytes
    byte[] meanByte = new byte[4];
    byte[] bByte = new byte[4];
    byte sign;
    // declare number of packets
    int totalPackets = 999;
    // declare nibblers
    int nb1,nb2;
    // declare reformed nibbles
    int reFnb1,reFnb2;
    // declare possition
    int pos = 4;
    // ras
    int x1 = 0,x2 = 0;
    // sde nn
    int mean,b,temp = 0;
    // byte array
    byte[] song = new byte[256*2*totalPackets];
    //arraylists
    ArrayList<Integer> sb = new ArrayList<Integer>();
    ArrayList<Integer> smpl = new ArrayList<Integer>();
    ArrayList<Integer> mns = new ArrayList<Integer>();
    ArrayList<Integer> bs = new ArrayList<Integer>();
    // initialize()
    for(int i = 1;i < totalPackets;i++){
        try{
            recieveSocket.receive(recievePacket);
            sign = (byte)( ( recieveBuffer[1] & 0x80) !=0 ? 0xff : 0x00);//if rxbuffer[1]&10000000=0 then sign =0 else =01111111 , we take
            meanByte[3] = sign;
            meanByte[2] = sign;
            meanByte[1] = recieveBuffer[1];
            meanByte[0] = recieveBuffer[0];
            mean = ByteBuffer.wrap(meanByte).order(ByteOrder.LITTLE_ENDIAN).getInt(); //convert the array into integer number using LITTLE_
            mns.add(mean);
            sign = (byte)( ( recieveBuffer[3] & 0x80) !=0 ? 0xff : 0x00);
            bByte[3] = sign;
            bByte[2] = sign;
            bByte[1] = recieveBuffer[3];
            bByte[0] = recieveBuffer[2];
            b = ByteBuffer.wrap(bByte).order(ByteOrder.LITTLE_ENDIAN).getInt();
            bs.add(b);

            for (int j = 4;j <= 131;j++){ //the remaining bytes are the samples
                nb1 = (int)(recieveBuffer[j] & 0x0000000F);
                nb2 = (int)((recieveBuffer[j] & 0x000000F0)>>4);
                reFnb1 = (nb2-8);
                sb.add(reFnb1);
                reFnb2 = (nb1-8);
                sb.add(reFnb2);
                reFnb1 = reFnb1*b;
                reFnb2 = reFnb2*b;
                x1 = temp + reFnb1 + mean;
                smpl.add(x1);
                x2 = reFnb1 + reFnb2 + mean;
                temp = reFnb2;
                smpl.add(x2);
                pos += 4;
                song[pos] = (byte)(x1 & 0x000000FF);
                song[pos + 1] = (byte)((x1 & 0x0000FF00)>>8);
                song[pos + 2] = (byte)(x2 & 0x000000FF);
                song[pos + 3] = (byte)((x2 & 0x0000FF00)>>8);


            }
        }catch (Exception ex){
            System.out.println(ex);
        }
    }
    if(mode==8){
        AudioFormat aqpcm = new AudioFormat(8000,16,1,true,false);
        SourceDataLine songStarts = AudioSystem.getSourceDataLine(aqpcm);
        songStarts.open(aqpcm,32000);
        songStarts.start();
        songStarts.write(song,0,256*2*totalPackets);
        songStarts.stop();
        songStarts.close();
        System.out.println("Song started");
    }
    // exporting subs
    saveAQDPCsub(audioCode,caseName,sb);
    // exporting samples
    saveAQDPCsamples(audioCode,caseName,smpl);
    // export means
    saveAQDPCmeans(audioCode,caseName,mns);
    // export betas
    saveAQDPCbetas(audioCode,caseName,bs);
    // disconecting
    recieveSocket.close();
    sendSocket.close();

}
```

```
// exports sub data for AQDPC
static void saveAQDPCsub(int audioCode,String caseName,ArrayList sb){
    BufferedWriter bufferedWriter = null;
    try{
        File file = new File("AQDPCM_SUBS_CODE_"+audioCode+"_"+caseName+".txt");
        if(!file.exists()){
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file,false);
        bufferedWriter = new BufferedWriter(fw);
        for(int i = 0 ; i < sb.size() ; i += 2){
            bufferedWriter.write("" + sb.get(i) + " " + sb.get(i+1));
            bufferedWriter.newLine();
        }

    }catch(IOException ioe){
        ioe.printStackTrace();
    }finally{
        try{
            if(bufferedWriter != null) bufferedWriter.close();
        }catch(Exception ex){

        }
    }
}
// exports samples data for AQDPC
static void saveAQDPCsamples(int audioCode,String caseName,ArrayList smpl){
    BufferedWriter mw = null;
    try{
        File file = new File("AQDPCM_SAMPLES_CODE_"+audioCode+"_"+caseName+".txt");
        if(!file.exists()){
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file,false);
        mw = new BufferedWriter(fw);
        for(int i = 0 ; i < smpl.size() ; i += 2){
            mw.write("" + smpl.get(i) + " " + smpl.get(i+1));
            mw.newLine();
        }

    }catch(IOException ioe){
        ioe.printStackTrace();
    }finally{
        try{
            if(mw != null) mw.close();
        }catch(Exception ex){
        }
    }
}
// exports means data for AQDPC
static void saveAQDPCmeans(int audioCode,String caseName,ArrayList mns){
    BufferedWriter pw = null;
    try{
        File f = new File("AQDPCM_MEANS_CODE_"+audioCode+"_"+caseName+".txt");
        if(!f.exists()){
            f.createNewFile();
        }
        FileWriter fw = new FileWriter(f,false);
        pw = new BufferedWriter(fw);
        for(int i = 0 ; i < mns.size() ; i += 2){
            pw.write("" + mns.get(i));
            pw.newLine();
        }
    }catch(IOException ioe){
        ioe.printStackTrace();
    }finally{
        try{
            if(pw != null) pw.close();
        }catch(Exception ex){
        }
    }
}
// exports betas data for AQDPC
static void saveAQDPCbetas(int audioCode,String caseName,ArrayList bs){
    BufferedWriter kw = null;
    try{
        File file = new File("AQDPCM_BETAS_CODE_"+audioCode+"_"+caseName+".txt");
        if(!file.exists()){
            file.createNewFile();
        }
        FileWriter fw = new FileWriter(file,false);
        kw = new BufferedWriter(fw);
        for(int i = 0 ; i < bs.size() ; i ++){
            kw.write("" + bs.get(i));
            kw.newLine();
        }

    }catch(IOException ioe){
        ioe.printStackTrace();
    }finally{
        try{
            if(kw != null) kw.close();
        }catch(Exception ex){

        }
    }
}
// User case 10
public static void vehicle_choice(int v_code,int server_listening_port,int client_listening_port,String code_pid) throws SocketException,IC
    // variable set for nameCase
    String nameCase="";
    String addT="";
    // list of output
    ArrayList<String> output = new ArrayList<String>();
    // variables for time
    double beginLoop=0;
    double finishloop=0;
    // initialize() inetaddress
    byte[] hostIP = { (byte)155,(byte)207,18,(byte)208 };
    InetAddress hostAddress = InetAddress.getByAddress(hostIP);
    // initialize() datagramsocket
    DatagramSocket sendSocket = new DatagramSocket();
    DatagramSocket recieveSocket = new DatagramSocket(client_listening_port);
    byte[] out_array = new byte[5000];
    DatagramPacket recievePacket = new DatagramPacket(out_array,out_array.length);
    recieveSocket.setSoTimeout(5000);
    // get currentTime()
    beginLoop = System.nanoTime();
    while(finishloop<4*60*1000){
        nameCase = "V"+Integer.toString(v_code)+"OBD=01 "+code_pid+"\r";
        byte[] code_arary = nameCase.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(code_arary,code_arary.length, hostAddress,server_listening_port);
        try{
            sendSocket.send(sendPacket);
            recieveSocket.receive(recievePacket);
            addT = new String(out_array,0,recievePacket.getLength());
            output.add(addT);
        }catch(Exception ex){
        }
        finishloop=(System.nanoTime()-beginLoop)/1000000;
```

```
        }
        //exports vehicle
        saveVehicleList(v_code,code_pid,output);
        //disconnecting
        recieveSocket.close();
        sendSocket.close();
    }
    // exports vehicle info
    public static void saveVehicleList(int vehicleCode,String pid,ArrayList output){
        BufferedWriter bufferedWriter = null;
        try{
            File f = new File("OBD_VEHICLE_"+vehicleCode+"_PID_"+pid+".txt");
            if(!f.exists()){
                f.createNewFile();
            }
            FileWriter fileWriter = new FileWriter(f,true);
            bufferedWriter = new BufferedWriter(fileWriter);
            for(int i = 0 ; i < output.size(); i++){
                bufferedWriter.write("" + output.get(i));
                bufferedWriter.newLine();
            }
        }catch(IOException ioe){
            ioe.printStackTrace();
        }finally{
            try{
                if(bufferedWriter != null) bufferedWriter.close();
            }catch(Exception ex){
            }
        }
    }


}
```