

STUDENT CODE

Programming Assignment 3: Streaming Video with RTSP and RTP

Classes

There are 4 classes in the assignment.

Client

Complete code

HINT: In particular pay attention to this line: `int RTSP_server_port = Integer.parseInt(argv[1]);`

```
// Client
// usage: java Client [Server hostname] [Server RTSP listening port]
// [Video file, movie.Mjpeg requested]

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Client{

    //GUI
    //----
    JFrame f = new JFrame("Client");
    JButton setupButton = new JButton("Setup");
    JButton playButton = new JButton("Play");
    JButton pauseButton = new JButton("Pause");
    JButton teardownButton = new JButton("Teardown");
    JPanel mainPanel = new JPanel();
    JPanel buttonPanel = new JPanel();
    JLabel iconLabel = new JLabel();
    ImageIcon icon;

    //RTP variables:
    //-----
    DatagramPacket rcvdp; //UDP packet received from the server
    DatagramSocket RTPsocket; //socket to be used to send and receive UDP
packets
    static int RTP_RCV_PORT = 25000; //port where the client will receive
the RTP packets

    Timer timer; //timer used to receive data from the UDP socket
    byte[] buf; //buffer used to store data received from the server

    //RTSP variables
    //-----
    //rtsp states
```

```
final static int INIT = 0;
final static int READY = 1;
final static int PLAYING = 2;
static int state; //RTSP state == INIT or READY or PLAYING
Socket RTSPsocket; //socket used to send/receive RTSP messages
//input and output stream filters
static BufferedReader RTSPBufferedReader;
static BufferedWriter RTSPBufferedWriter;
static String VideoFileName; //video file to request to the server
int RTSPSeqNb = 0; //Sequence number of RTSP messages within the
session
int RTSPid = 0; //ID of the RTSP session (given by the RTSP Server)

final static String CRLF = "\r\n";

//Video constants:
//-----
static int MJPEG_TYPE = 26; //RTP payload type for MJPEG video

//-----
//Constructor
//-----
public Client() {

    //build GUI
    //-----

    //Frame
    f.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });

    //Buttons
    buttonPanel.setLayout(new GridLayout(1,0));
    buttonPanel.add(setupButton);
    buttonPanel.add(playButton);
    buttonPanel.add(pauseButton);
    buttonPanel.add(tearButton);
    setupButton.addActionListener(new setupButtonListener());
    playButton.addActionListener(new playButtonListener());
    pauseButton.addActionListener(new pauseButtonListener());
    tearButton.addActionListener(new tearButtonListener());

    //Image display label
    iconLabel.setIcon(null);

    //frame layout
    mainPanel.setLayout(null);
    mainPanel.add(iconLabel);
    mainPanel.add(buttonPanel);
    iconLabel.setBounds(0,0,380,280);
    buttonPanel.setBounds(0,280,380,50);

    f.getContentPane().add(mainPanel, BorderLayout.CENTER);
    f.setSize(new Dimension(390,370));
```

```
f.setVisible(true);

//init timer
//-----
timer = new Timer(20, new timerListener());
timer.setInitialDelay(0);
timer.setCoalesce(true);

//allocate enough memory for the buffer used to receive data from
the server
buf = new byte[15000];
}

//-----
//main
//-----
public static void main(String argv[]) throws Exception
{
    //Create a Client object
    Client theClient = new Client();

    //get server RTSP port and IP address from the command line
    //-----
    int RTSP_server_port = Integer.parseInt(argv[1]);
    String ServerHost = argv[0];
    InetAddress ServerIPAddr = InetAddress.getByName(ServerHost);

    //get video filename to request:
    VideoFileName = argv[2];

    //Establish a TCP connection with the server to exchange RTSP
messages
    //-----
    theClient.RTSPsocket = new Socket(ServerIPAddr, RTSP_server_port);

    //Set input and output stream filters:
    RTSPBufferedReader = new BufferedReader(new
InputStreamReader(theClient.RTSPsocket.getInputStream() ));
    RTSPBufferedWriter = new BufferedWriter(new
OutputStreamWriter(theClient.RTSPsocket.getOutputStream() ));

    //init RTSP state:
    state = INIT;
}

//-----
//Handler for buttons
//-----

//TO COMPLETE

//Handler for Setup button
//-----
class setupButtonListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
```

```
        //System.out.println("Setup Button pressed !");

        if (state == INIT)
        {
            //Init non-blocking RTPsocket that will be used to receive
data
            try{
                //construct a new DatagramSocket to receive RTP packets from
the server, on port RTP_RCV_PORT
                //RTPsocket = ...

                //set Timeout value of the socket to 5msec.
                //.....

            }
            catch (SocketException se)
            {
                System.out.println("Socket exception: "+se);
                System.exit(0);
            }

            //init RTSP sequence number
            RTSPSeqNb = 1;

            //Send SETUP message to the server
            send_RTSP_request("SETUP");

            //Wait for the response
            if (parse_server_response() != 200)
                System.out.println("Invalid Server Response");
            else
            {
                //change RTSP state and print new state
                //state = ....
                //System.out.println("New RTSP state: ....");
            }
        } //else if state != INIT then do nothing
    }

}

//Handler for Play button
//-----
class playButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){

        //System.out.println("Play Button pressed !");

        if (state == READY)
        {
            //increase RTSP sequence number
            //.....

            //Send PLAY message to the server
            send_RTSP_request("PLAY");

            //Wait for the response
```

```
        if (parse_server_response() != 200)
            System.out.println("Invalid Server Response");
        else
        {
            //change RTSP state and print out new state
            //.....
            // System.out.println("New RTSP state: ...")

            //start the timer
            timer.start();
        }
    } //else if state != READY then do nothing
}

//Handler for Pause button
//-----
class pauseButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){

        //System.out.println("Pause Button pressed !");

        if (state == PLAYING)
        {
            //increase RTSP sequence number
            //.....

            //Send PAUSE message to the server
            send_RTSP_request("PAUSE");

            //Wait for the response
            if (parse_server_response() != 200)
                System.out.println("Invalid Server Response");
            else
            {
                //change RTSP state and print out new state
                //.....
                //System.out.println("New RTSP state: ...");

                //stop the timer
                timer.stop();
            }
        }
        //else if state != PLAYING then do nothing
    }
}

//Handler for Teardown button
//-----
class teardownButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e){

        //System.out.println("Teardown Button pressed !");

        //increase RTSP sequence number
        // .....
```

```
//Send TEARDOWN message to the server
send_RTSP_request("_____");

//Wait for the response
if (parse_server_response() != 200)
    System.out.println("Invalid Server Response");
else
{
    //change RTSP state and print out new state
    //.....
    //System.out.println("New RTSP state: ...");

    //stop the timer
    timer.stop();

    //exit
    System.exit(0);
}
}

//-----
//Handler for timer
//-----

class timerListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {

        //Construct a DatagramPacket to receive data from the UDP socket
        rcvdp = new DatagramPacket(buf, buf.length);

        try{
            //receive the DP from the socket:
            RTPsocket.receive(rcvdp);

            //create an RTPpacket object from the DP
            RTPpacket rtp_packet = new RTPpacket(rcvdp.getData(),
rcvdp.getLength());

            //print important header fields of the RTP packet received:
            System.out.println("Got RTP packet with SeqNum #
"+rtp_packet.getsequencenumber()+" TimeStamp
"+rtp_packet.gettimestamp()+" ms, of type
"+rtp_packet.getpayloadtype());

            //print header bitstream:
            rtp_packet.printhead();

            //get the payload bitstream from the RTPpacket object
            int payload_length = rtp_packet.getpayload_length();
            byte [] payload = new byte[payload_length];
            rtp_packet.getpayload(payload);

            //get an Image object from the payload bitstream
```

```
Toolkit toolkit = Toolkit.getDefaultToolkit();
Image image = toolkit.createImage(payload, 0, payload_length);

//display the image as an ImageIcon object
icon = new ImageIcon(image);
iconLabel.setIcon(icon);
}
catch (InterruptedException iioe){
    //System.out.println("Nothing to read");
}
catch (IOException ioe) {
    System.out.println("Exception caught: "+ioe);
}
}
}

//-----
//Parse Server Response
//-----
private int parse_server_response()
{
    int reply_code = 0;

    try{
        //parse status line and extract the reply_code:
        String StatusLine = RTSPBufferedReader.readLine();
        //System.out.println("RTSP Client - Received from Server:");
        System.out.println(StatusLine);

        StringTokenizer tokens = new StringTokenizer(StatusLine);
        tokens.nextToken(); //skip over the RTSP version
        reply_code = Integer.parseInt(tokens.nextToken());

        //if reply code is OK get and print the 2 other lines
        if (reply_code == _____)
        {
            String SeqNumLine = RTSPBufferedReader.readLine();
            System.out.println(SeqNumLine);

            String SessionLine = RTSPBufferedReader.readLine();
            System.out.println(SessionLine);

            //if state == INIT gets the Session Id from the SessionLine
            tokens = new StringTokenizer(SessionLine);
            tokens.nextToken(); //skip over the Session:
            RTSPid = Integer.parseInt(tokens.nextToken());
        }
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
    }

    return(_____);
}
```

```
//-----  
//Send RTSP Request  
//-----  
  
//TO COMPLETE  
  
private void send_RTSP_request(String request_type)  
{  
    try{  
        //Use the RTSPBufferedWriter to write to the RTSP socket  
  
        //write the request line:  
        //RTSPBufferedWriter.write(...);  
  
        //write the CSeq line:  
        //.....  
  
        //check if request_type is equal to "SETUP" and in this case  
        write the Transport: line advertising to the server the port used to  
        receive the RTP packets RTP_RCV_PORT  
        //if ....  
        //otherwise, write the Session line from the RTSPid field  
        //else ....  
  
        RTSPBufferedWriter.flush();  
    }  
    catch(Exception ex)  
    {  
        System.out.println("Exception caught: "+ex);  
        System.exit(0);  
    }  
}  
  
} //end of Class Client
```

Server

Complete code

HINT: you need to pay attention to certain lines like `int RTSPport = Integer.parseInt(argv[0]);`

```
// Server
// usage: java Server [RTSP listening port]

import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.Timer;

public class Server extends JFrame implements ActionListener {

    //RTP variables:
    //-----
    DatagramSocket RTPsocket; //socket to be used to send and receive UDP
packets
    DatagramPacket senddp; //UDP packet containing the video frames

    InetAddress ClientIPAddr; //Client IP address
    int RTP_dest_port = 0; //destination port for RTP packets (given by
the RTSP Client)

    //GUI:
    //-----
    JLabel label;

    //Video variables:
    //-----
    int imagenb = 0; //image nb of the image currently transmitted
    VideoStream video; //VideoStream object used to access video frames
    static int MJPEG_TYPE = 26; //RTP payload type for MJPEG video
    static int FRAME_PERIOD = 100; //Frame period of the video to stream,
in ms
    static int VIDEO_LENGTH = 500; //length of the video in frames

    Timer timer; //timer used to send the images at the video frame rate
    byte[] buf; //buffer used to store the images to send to the client

    //RTSP variables
    //-----
    //rtsp states
    final static int INIT = 0;
    final static int READY = 1;
    final static int PLAYING = 2;
    //rtsp message types
    final static int SETUP = 3;
    final static int PLAY = 4;
    final static int PAUSE = 5;
```

```
final static int TEARDOWN = 6;

static int state; //RTSP Server state == INIT or READY or PLAY
Socket RTSPsocket; //socket used to send/receive RTSP messages
//input and output stream filters
static BufferedReader RTSPBufferedReader;
static BufferedWriter RTSPBufferedWriter;
static String VideoFileName; //video file requested from the client
static int RTSP_ID = 123456; //ID of the RTSP session
int RTSPSeqNb = 0; //Sequence number of RTSP messages within the
session

final static String CRLF = "\r\n";

//-----
//Constructor
//-----
public Server(){

    //init Frame
    super("Server");

    //init Timer
    timer = _____ Timer(FRAME_PERIOD, this);
    _____ .setInitialDelay(0);
    timer.setCoalesce(true);

    //allocate memory for the sending buffer
    buf = new byte[15000];

    //Handler to close the main window
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            //stop the timer and exit
            timer.stop();
            System.exit(0);
        }
    });

    //GUI:
    label = new JLabel("Send frame #          ", JLabel.CENTER);
    getContentPane().add(label, BorderLayout.CENTER);
}

//-----
//main
//-----
public static void main(String argv[]) throws Exception
{
    //create a Server object
    Server theServer = _____

    //show GUI:
    theServer.pack();
    theServer.setVisible(true);

    //get RTSP socket port from the command line
    int RTSPport = Integer.parseInt(argv[0]);
}
```

```
//Initiate TCP connection with the client for the RTSP session
ServerSocket listenSocket = new ServerSocket(RTSPport);
theServer.RTSPsocket = listenSocket.accept();
listenSocket.close();

//Get Client IP address
theServer.ClientIPAddr = theServer.RTSPsocket.getInetAddress();

//Initiate RTSPstate
state = INIT;

//Set input and output stream filters:
RTSPBufferedReader = new BufferedReader(new
InputStreamReader(theServer.RTSPsocket.getInputStream() ) );
RTSPBufferedWriter = new BufferedWriter(new
OutputStreamWriter(theServer.RTSPsocket.getOutputStream() ) );

//Wait for the SETUP message from the client
int request_type;
boolean done = _____;
while(!done)
{
    request_type = theServer.parse_RTSP_request(); //blocking

    if (request_type == SETUP)
    {
        done = true;

        //update RTSP state
        state = _____;
        System.out.println("New RTSP state: READY");

        //Send response
        theServer.send_RTSP_response();

        //init the VideoStream object:
        _____ .video = new VideoStream(VideoFileName);

        //init RTP socket
        theServer._____ = new DatagramSocket();
    }
}

//loop to handle RTSP requests
while(true)
{
    //parse the request
    request_type = theServer.parse_RTSP_request(); //blocking

    if ((request_type == PLAY) && (state == READY))
    {
        //send back response
        theServer.send_RTSP_response();
        //start timer
        theServer.timer.start();
        //update state
    }
}
```

```
        state = PLAYING;
        System.out.println("New RTSP state: PLAYING");
    }
    else if ((request_type == PAUSE) && (state == PLAYING))
    {
        //send back response
        theServer.send_RTSP_response();
        //stop timer
        theServer.timer.stop();
        //update state
        state = READY;
        System.out.println("New RTSP state: READY");
    }
    else if (request_type == TEARDOWN)
    {
        //send back response
        theServer.send_RTSP_response();
        //stop timer
        theServer.timer.stop();
        //close sockets
        theServer.RTSPsocket.close();
        theServer.RTPsocket.close();

        System.exit(0);
    }
}

//-----
//Handler for timer
//-----
public void actionPerformed(ActionEvent e) {

    //if the current image nb is less than the length of the video
    if (imagenb < VIDEO_LENGTH)
    {
        //update current imagenb
        imagenb++;

        try {
            //get next frame to send from the video, as well as its size
            int image_length = video.getnextframe(buf);

            //Builds an RTPpacket object containing the frame
            RTPpacket rtp_packet = new RTPpacket(MJPEG_TYPE, imagenb,
imagenb*FRAME_PERIOD, buf, image_length);

            //get to total length of the full rtp packet to send
            int packet_length = rtp_packet.getlength();

            //retrieve the packet bitstream and store it in an array of
bytes
            byte[] packet_bits = new byte[packet_length];
            rtp_packet.getpacket(packet_bits);

            //send the packet as a DatagramPacket over the UDP socket
```

```
        senddp = new DatagramPacket(packet_bits, packet_length,
ClientIPAddr, RTP_dest_port);
        RTPsocket.send(senddp);

        //System.out.println("Send frame #" + imagenb);
        //print the header bitstream
        rtp_packet.printhead();

        //update GUI
        label.setText("Send frame #" + imagenb);
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: " + ex);
        System.exit(0);
    }
}
else
{
    //if we have reached the end of the video file, stop the timer
    timer.stop();
}
}

//-----
//Parse RTSP Request
//-----
private int parse_RTSP_request()
{
    int request_type = -1;
    try{
        //parse request line and extract the request_type:
        String RequestLine = RTSPBufferedReader.readLine();
        //System.out.println("RTSP Server - Received from Client:");
        System.out.println(RequestLine);

        StringTokenizer tokens = new StringTokenizer(RequestLine);
        String request_type_string = tokens.nextToken();

        //convert to request_type structure:
        if ((new String(request_type_string)).compareTo("SETUP") == 0)
            request_type = SETUP;
        else if ((new String(request_type_string)).compareTo("PLAY") ==
0)
            request_type = _____;
        else if ((new String(request_type_string)).compareTo("PAUSE") ==
0)
            request_type = _____;
        else if ((new String(request_type_string)).compareTo("TEARDOWN")
== 0)
            request_type = _____;

        if (request_type == SETUP)
        {
            //extract VideoFileName from RequestLine
            VideoFileName = tokens.nextToken();
        }
    }
}
```

```
//parse the SeqNumLine and extract CSeq field
String SeqNumLine = RTSPBufferedReader.readLine();
System.out.println(SeqNumLine);
tokens = new StringTokenizer(SeqNumLine);
tokens.nextToken();
RTSPSeqNb = Integer.parseInt(tokens.nextToken());

//get LastLine
String LastLine = RTSPBufferedReader.readLine();
System.out.println(LastLine);

if (request_type == SETUP)
{
    //extract RTP_dest_port from LastLine
    tokens = new StringTokenizer(LastLine);
    for (int i=0; i<3; i++)
        tokens.nextToken(); //skip unused stuff
    RTP_dest_port = Integer.parseInt(tokens.nextToken());
}
//else LastLine will be the SessionId line ... do not check for
now.
}
catch(Exception ex)
{
    System.out.println("Exception caught: "+ex);
    System.exit(0);
}
return(request_type);
}

//-----
//Send RTSP Response
//-----
private void send_RTSP_response()
{
    try{
        RTSPBufferedWriter.write("RTSP/1.0 _____ OK"+_____);
        RTSPBufferedWriter.write("CSeq: "+RTSPSeqNb+CRLF);
        RTSPBufferedWriter.write("Session: "+RTSP_ID+CRLF);
        RTSPBufferedWriter.flush();
        //System.out.println("RTSP Server - Sent response to Client.");
    }
    catch(Exception ex)
    {
        System.out.println("Exception caught: "+ex);
        System.exit(0);
    }
}
} // end of Server class
```

RTPpacket.java

Complete code

```
//class RTPpacket

public class RTPpacket{

    //size of the RTP header:
    static int HEADER_SIZE = 12;

    //Fields that compose the RTP header
    public int Version;
    public int Padding;
    public int Extension;
    public int CC;
    public int Marker;
    public int PayloadType;
    public int SequenceNumber;
    public int TimeStamp;
    public int Ssrc;

    //Bitstream of the RTP header
    public byte[] header;

    //size of the RTP payload
    public int payload_size;
    //Bitstream of the RTP payload
    public byte[] payload;

    //-----
    //Constructor of an RTPpacket object from header fields and payload
    bitstream
    //-----
    public RTPpacket(int PType, int Framenb, int Time, byte[] data, int
    data_length){
        //fill by default header fields:
        Version = 2;
        Padding = 0;
        Extension = 0;
        CC = 0;
        Marker = 0;
        Ssrc = 0;

        //fill changing header fields:
        SequenceNumber = Framenb;
        TimeStamp = Time;
        PayloadType = PType;

        //build the header bistream:
        //-----
        header = new byte[HEADER_SIZE];

        //TO COMPLETE
```

```
//fill the header array of byte with RTP header fields

//header[0] = ...
// .....

//the payload bitstream:
payload_size = data_length;
payload = new byte[data_length];

//fill payload array of byte from data (given in parameter of the
constructor)
//.....
}

//-----
//Constructor of an RTPpacket object from the packet bistream
//-----
public RTPpacket(byte[] packet, int packet_size)
{
    // default fields:
    Version = 2;
    Padding = 0;
    Extension = 0;
    CC = 0;
    Marker = 0;
    Ssrc = 0;

    //check if total packet size is lower than the header size
    if (packet_size >= HEADER_SIZE)
    {
        //get the header bitsream:
        header = new byte[HEADER_SIZE];
        for (int i=0; i < HEADER_SIZE; i++)
            header[i] = packet[i];

        //get the payload bitstream:
        payload_size = packet_size - HEADER_SIZE;
        payload = new byte[payload_size];
        for (int i=HEADER_SIZE; i < packet_size; i++)
            payload[i-HEADER_SIZE] = packet[i];

        //interpret the changing fields of the header:
        PayloadType = header[1] & 127;
        SequenceNumber = unsigned_int(header[3]) +
256*unsigned_int(header[2]);
        TimeStamp = unsigned_int(header[7]) +
256*unsigned_int(header[6]) + 65536*unsigned_int(header[5]) +
16777216*unsigned_int(header[4]);
    }
}

//-----
//getpayload: return the payload bistream of the RTPpacket and its
size
//-----
public int getpayload(byte[] data) {
```

```
        for (int i=0; i < payload_size; i++)
            data[i] = payload[i];

        return(payload_size);
    }

    //-----
    //getpayload_length: return the length of the payload
    //-----
    public int getpayload_length() {
        return(payload_size);
    }

    //-----
    //getlength: return the total length of the RTP packet
    //-----
    public int getlength() {
        return(payload_size + HEADER_SIZE);
    }

    //-----
    //getpacket: returns the packet bitstream and its length
    //-----
    public int getpacket(byte[] packet)
    {
        //construct the packet = header + payload
        for (int i=0; i < HEADER_SIZE; i++)
            packet[i] = header[i];
        for (int i=0; i < payload_size; i++)
            packet[i+HEADER_SIZE] = payload[i];

        //return total size of the packet
        return(payload_size + HEADER_SIZE);
    }

    //-----
    //gettimestamp
    //-----
    public int gettimestamp() {
        return(TimeStamp);
    }

    //-----
    //getsequencenumber
    //-----
    public int getsequencenumber() {
        return(SequenceNumber);
    }

    //-----
    //getpayloadtype
    //-----
    public int getpayloadtype() {
        return(PayloadType);
    }
}
```

```
//-----  
//print headers without the SSRC  
//-----  
public void printhead()  
{  
    //TO DO: uncomment  
    /*  
    for (int i=0; i < (HEADER_SIZE-4); i++)  
        {  
            for (int j = 7; j>=0 ; j--)  
                if (((1<<j) & header[i] ) != 0)  
                    System.out.print("1");  
                else  
                    System.out.print("0");  
                System.out.print(" ");  
            }  
  
        System.out.println();  
        */  
}  
  
//return the unsigned value of 8-bit integer nb  
static int unsigned_int(int nb) {  
    if (nb >= 0)  
        return(nb);  
    else  
        return(256+nb);  
}  
} // end of RTPpacket Class
```

VideoStream

Complete code

This class is used to read video data from the file on disk.

```
//VideoStream

import java.io.*;

public class VideoStream {

    FileInputStream fis; //video file
    int frame_nb; //current frame nb

    //-----
    //constructor
    //-----
    public VideoStream(String filename) throws Exception{

        //init variables
        fis = new FileInputStream(_____);
        frame_nb = 0;
    }

    //-----
    // getnextframe
    //returns the next frame as an array of byte and the size of the
frame
    //-----
    public int getnextframe(byte[] frame) throws Exception
    {
        int length = 0;
        String length_string;
        byte[] frame_length = new byte[5];

        //read current frame length
        fis.read(frame_length,_____,_____);

        //transform frame_length to integer
        length_string = new String(_____);
        length = Integer.parseInt(_____);

        return(fis.read(frame,0,length));
    }
} // end of VideoStream class
```