**Project 02**

**Name:** Jose Juan Sandoval

**Link to Project:** <https://github.com/Juanchiselo/CS380/tree/master/Projects/Project%2002>

**Java Code**

PhysLayerClient.java

package Project02;  
  
import java.io.IOException;  
import java.net.Socket;  
import java.net.UnknownHostException;  
  
public class PhysLayerClient  
{  
 private static Socket *socket*;  
  
 public static void main(String[] args)  
 {  
 *connectToServer*();  
 }  
  
 */\*\*  
 \* Connects the client to the server and  
 \* creates a Listener thread.  
 \*/* public static void connectToServer()  
 {  
 String hostName = "codebank.xyz";  
 int portNumber = 38002;  
  
 try  
 {  
 *socket* = new Socket(hostName, portNumber);  
 new ListenerThread(*socket*).start();  
 System.*out*.println("Connected to server.");  
 }  
 catch (UnknownHostException e)  
 {  
 System.*err*.println("ERROR: Unknown host " + hostName + ".");  
 }  
 catch (Exception e)  
 {  
 System.*err*.println("ERROR: Could not connect to " + hostName + ".");  
 }  
 }  
  
 */\*\*  
 \* Disconnects the client from the server.  
 \*/* public static void disconnectFromServer()  
 {  
 try  
 {  
 *socket*.close();  
 System.*out*.println("Disconnected from server.");  
 }  
 catch (IOException e)  
 {  
 System.*err*.println("ERROR: " + e.getMessage());  
 }  
 }  
}

ListenerThread.java

package Project02;  
  
import java.io.\*;  
import java.net.Socket;  
import java.util.HashMap;  
  
public class ListenerThread extends Thread  
{  
 public volatile static boolean *endThread* = false;  
 private Socket socket = null;  
  
 public ListenerThread(Socket socket)  
 {  
 super("Listener Thread");  
 this.socket = socket;  
 }  
  
 */\*\*  
 \* The overridden run() function belonging to the Thread class.  
 \* This is what handles the communication between the server and the client.  
 \*/* public void run()  
 {  
 final int PREAMBLE\_SIZE = 64;  
 final int DATA\_SIZE = 32;  
 byte[] data;  
  
 float baseline = calculateBaseline(PREAMBLE\_SIZE);  
 System.*out*.println("Baseline established from preamble: " + baseline);  
  
 System.*out*.print("Received " + DATA\_SIZE + " bytes: ");  
 data = receiveData(DATA\_SIZE, baseline);  
 System.*out*.println();  
  
 respondToServer(data);  
 PhysLayerClient.*disconnectFromServer*();  
 }  
  
 */\*\*  
 \* Reads the preamble and calculates the baseline based  
 \* on an average of the received high and low signals.  
 \** ***@param*** *PREAMBLE\_SIZE - The size of the preamble.  
 \** ***@return*** *- Returns the baseline.  
 \*/* private float calculateBaseline(final int PREAMBLE\_SIZE)  
 {  
 float baseline = 0.0f;  
 try  
 {  
 for(int i = 0; i < PREAMBLE\_SIZE; i++)  
 baseline += socket.getInputStream().read();  
 baseline /= PREAMBLE\_SIZE;  
 }  
 catch (Exception e)  
 {  
 System.*err*.println("ERROR: " + e.getMessage());  
 }  
  
 return baseline;  
 }  
  
 */\*\*  
 \* Creates the HashMap containing the 4B/5B conversion table.  
 \** ***@return*** *- Returns the 4B/5B conversion table HashMap.  
 \*/* private HashMap<String, String> create4B5BConversionTable()  
 {  
 HashMap<String, String> hashMap = new HashMap<>();  
  
 hashMap.put("11110", "0000");  
 hashMap.put("01001", "0001");  
 hashMap.put("10100", "0010");  
 hashMap.put("10101", "0011");  
 hashMap.put("01010", "0100");  
 hashMap.put("01011", "0101");  
 hashMap.put("01110", "0110");  
 hashMap.put("01111", "0111");  
 hashMap.put("10010", "1000");  
 hashMap.put("10011", "1001");  
 hashMap.put("10110", "1010");  
 hashMap.put("10111", "1011");  
 hashMap.put("11010", "1100");  
 hashMap.put("11011", "1101");  
 hashMap.put("11100", "1110");  
 hashMap.put("11101", "1111");  
  
 return hashMap;  
 }  
  
 */\*\*  
 \* Receives and decodes the data.  
 \** ***@param*** *DATA\_SIZE - The size of the data to be received.  
 \** ***@param*** *baseline - The baseline.  
 \** ***@return*** *- Returns the data in binary as strings.  
 \*/* private byte[] receiveData(final int DATA\_SIZE, float baseline)  
 {  
 // The size is multiplied by 2 because  
 // we are receiving the bytes in halves.  
 String[] receivedData = new String[DATA\_SIZE \* 2];  
 byte[] data = new byte[DATA\_SIZE];  
 try  
 {  
 boolean previousSignal = false;  
 boolean currentSignal;  
 String fiveBits;  
 HashMap<String, String> fourBfiveBTable = create4B5BConversionTable();  
  
 for(int i = 0; i < receivedData.length; i++)  
 {  
 fiveBits = "";  
 for(int j = 0; j < 5; j++)  
 {  
 currentSignal = socket.getInputStream().read() > baseline;  
  
 // This is where the decoding of the NRZI signal occurs.  
 // The current signal is compared with the baseline.  
 // If the signal is below the baseline it is a 0.  
 // if the signal is above the baseline it is a 1.  
 // The current and previous signals get compared.  
 // If they are the same the data bit is a 0,  
 // if they are different the data bit is a 1.  
 if (previousSignal == currentSignal)  
 fiveBits += "0";  
 else  
 fiveBits += "1";  
  
 previousSignal = currentSignal;  
 }  
 // The 5B to 4B conversion occurs here.  
 receivedData[i] = fourBfiveBTable.get(fiveBits);  
 }  
  
 String firstNibble;  
 String secondNibble;  
 String completeByteString;  
 int completeByte;  
  
 // Data reconstruction occurs here.  
 for(int i = 0, j = 0; i < data.length; i++, j += 2)  
 {  
 firstNibble = receivedData[j];  
 secondNibble = receivedData[j + 1];  
 completeByteString = firstNibble + secondNibble;  
 completeByte = Integer.*parseInt*(completeByteString, 2);  
 System.*out*.print(Integer.*toHexString*(completeByte).toUpperCase());  
 data[i] = (byte) completeByte;  
 }  
 }  
 catch(Exception e)  
 {  
 System.*err*.println("ERROR: " + e.getMessage());  
 }  
  
 return data;  
 }  
  
 */\*\*  
 \* Sends the response to the server.  
 \** ***@param*** *response - The response to be sent to the server.  
 \*/* private void respondToServer(byte[] response)  
 {  
 try  
 {  
 socket.getOutputStream().write(response);  
  
 int serverResponse;  
 if((serverResponse = socket.getInputStream().read()) == 1)  
 System.*out*.println("Response good.");  
 else  
 System.*out*.println("Bad response. Server returned " + serverResponse);  
 }  
 catch (IOException e)  
 {  
 System.*err*.println("ERROR: " + e.getMessage());  
 }  
 }  
}