Client class

**package** actualProjectPackage;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** ClientClass {

**private** **int** requestId;//variable for request id

**private** **int** measurementId;//variable for measurement id

**private** **static** String *requestMessage*;//string object for request message

**private** **static** **byte**[] *messageByte*;

**private** **final** **static** **long** *C* = 7919;//C value for checksum

**private** **final** **static** **long** *D* = 65536;//D value for checksum

**private** **static** **int** *groupNo*;//variable for group number used in regular expressions

**private** **static** String *toMatchStr*;

**private** **static** String *patternTo*;

**public** ClientClass(**int** aRequestId,**int** aMeasurementId)

{requestId = aRequestId;

measurementId = aMeasurementId;

*requestMessage* = "";

*messageByte* = **null**;

*groupNo* = 0;

*toMatchStr* = **null**;

*patternTo* = **null**;

}//initializer

//method to create request message

**public** String toString()

{**return** "<request>" + "\n" + "\t"+ "<id>" + requestId + "</id>" + "\n" + "\t" + "<measurement>" + measurementId +"</measurement>" +"\n" + "</request>";

}//toString() class

//method to convert string to byte[]

**public** **static** **byte**[] convertStringtoByteArray(String aRequestMessage)

{*requestMessage* = aRequestMessage;

**return** *requestMessage*.getBytes();}//convertStringtoByteArray class

//method to calculate length of a string

**public** **int** calculateLength(String aRequestMessage)

{*requestMessage* = aRequestMessage;

**return** *requestMessage*.length();}//calculateLength class

//method to calculate CheckSum over byte[]

**public** **static** **long** calculateCheckSum(**byte**[] aMessageByte)

{*messageByte* = aMessageByte;

**long** S =0;

**long** index;

**if** (*messageByte*.length % 2 == 0)//if character sequence/byte[] (using UTF-8 encoding) is even

{**long**[] unsignedWords = **new** **long**[*messageByte*.length/2];//create array of 16 bit unsigned words

**int** k = 0;

**for**(**int** i=0;i<*messageByte*.length;i = i+2)

{ **long** x = (**long**)(((*messageByte*[i] & 0xFF) << 8) | (*messageByte*[i+1] & 0xFF));//bit wise shift and or to form unsigned word.lower index arrives first

unsignedWords[k] = x; k = k +1;}

**for** (**int** j = 0; j<unsignedWords.length;j++)//calculating checksum over array of 16 bit words

{index = S ^ unsignedWords[j];

S = ((*C*\*index) % *D*);}

}

//else if character sequence is odd in length

**else** **if** (*messageByte*.length % 2 !=0)

{**long** [] unsignedWords = **new** **long** [(*messageByte*.length+1)/2];

**int** o = 0;

**for** (**int** m=0;m<*messageByte*.length-1;m = m + 2)//iterating over except last odd element

{**long** x = (**long**)(((*messageByte*[m] & 0xFF) << 8) | (*messageByte*[m+1] & 0xFF));

unsignedWords[o] = x;o = o + 1;}

unsignedWords[(*messageByte*.length+1)/2 -1 ] = (**long**)(((*messageByte*[*messageByte*.length-1] & 0xFF) << 8) | (0 & 0xFF));//calculating unsigned word for last odd element

**for** (**int** n = 0; n<unsignedWords.length;n++)

{index = S ^ unsignedWords[n];

S = ((*C*\*index) % *D*);}

}

**return** S;

}//calculateCheckSum class

//method to extract & return string objects using regular expressions using group number of required string

//, string in which the matching is to take place, pattern to be matched/looked for

**public** **static** String getPatternMatch(**int** aGroupNo,String aToMatchStr,String aPatternTo)

{

*groupNo* = aGroupNo;

*toMatchStr* = aToMatchStr;

*patternTo* = aPatternTo;

Pattern pat = Pattern.*compile*(*patternTo*);

Matcher mat = pat.matcher(*toMatchStr*);

mat.find();

**return** mat.group(*groupNo*);//return required string

}//class getPatternMatch

}//class ClientClass

Client main class

**package** actualProjectPackage;

**import** java.io.\*;

**import** java.net.\*;

**import** java.nio.charset.StandardCharsets;

**import** java.util.\*;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** ClientMain {

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

// **TODO** Auto-generated method stub

**while**(**true**){//do this forever until application is terminated

String serverMessage;

Scanner inp = **new** Scanner(System.*in*);

BufferedReader bufread = **null**;

**final** **int** INIT\_TIMEOUT\_VAL = 1000;

DatagramSocket clientSocket = **null**;

**int** counter=1;

//reading data file to randomly pick measurement id

**try** {

bufread = **new** BufferedReader(**new** FileReader("C:\\Users\\comp\\Downloads\\data.txt"));//reading data file

} **catch** (FileNotFoundException e) {System.*out*.println("File not found");}

String measureid[] = **new** String[100];//initializing an array to store measurement values

**int** i =0;

String line ;

**while** ((line = bufread.readLine()) != **null**)

{String info[] = line.split("\\s+");

measureid[i] = info[0]; //creating array to randomly pick measurement values from data file

i++;

}

Random rand = **new** Random();

**int** MeasId = Integer.*parseInt*(measureid[rand.nextInt(measureid.length)]);//picking random measure id and converting it to integer

//MeasId =1234;

outerloop:

//loop for step 5, do this i.e. re-send same measurement value as long as error codes are being received

**while**(**true**){Thread.*sleep*(500);

//loop for step 4, do this i.e. re-send same measurement value if integrity checksum of response message fail

**while**(**true**)

{Thread.*sleep*(500);

**int** randRequest = rand.nextInt(1 + Short.*MAX\_VALUE* - Short.*MIN\_VALUE*);//generating random 16 bit unsigned request id within unsigned short data limit

ClientClass aObj = **new** ClientClass(randRequest,MeasId);

String message = aObj.toString();//creating request message

String formattedMessage = message.replaceAll("\\s+","");//removing all white spaces/tabs before transmission

**byte**[] forCheckSum = ClientClass.*convertStringtoByteArray*(formattedMessage); //converting request message to byte[] for checksum

**long** checkSum = ClientClass.*calculateCheckSum*(forCheckSum) ; //calculating checksum over byte[] which in turn contain the ASCII code for each character

String actualMessage = formattedMessage + "" + checkSum;//appending check sum to request message

System.*out*.println(aObj.toString() + "\n"+checkSum+"\n");

**byte** [] toSendBytes = ClientClass.*convertStringtoByteArray*(actualMessage);//converting to byte[] from transmission of entire request message including checksum

InetAddress serverAddress = InetAddress.*getLocalHost*();//assuming server to be same as client

DatagramPacket toSendPacket = **new** DatagramPacket(toSendBytes,toSendBytes.length,serverAddress,9999);//packet to send request

**try** {

clientSocket = **new** DatagramSocket();//choosing arbitrary socket to send request

} **catch** (SocketException e1) {

System.*out*.println("No socket available");

}

//In the following lines, application attempts to send & receive packets,on failure the timeout is doubled till 4th timeout

**int** varTimeoutVal = INIT\_TIMEOUT\_VAL;

**byte**[] buffer = **new** **byte**[1000];//defining large enough buffer byte[] to receive response message from server

DatagramPacket toReceivePacket = **new** DatagramPacket(buffer,buffer.length);

**do**

{**try**

{

**try** {

clientSocket.send(toSendPacket);

} **catch** (IOException e) {

System.*out*.println("Unable to send the packets" + e);

}//sending message request

clientSocket.setSoTimeout(varTimeoutVal);//setting timeout interval

System.*out*.println("Receiving response from server: Attempt " +"#"+counter);

clientSocket.receive(toReceivePacket);//trying to receive packets

**break**;//breaking off after receiving packet successfully

}

**catch**(InterruptedIOException e)//catching timeout exception

{counter = counter + 1;varTimeoutVal = 2\*varTimeoutVal; //increasing counter & doubling timeout value

**if** (counter == 5)

{System.*err*.println("ERROR: Communication failure");System.*exit*(0);} //error message after 4th timeout

}

}**while**(counter <=4);//repeat sending packet until 4th timeout or if a packet is received

//in the following lines , application processes response message upon successful reception

**byte** [] serverData = **new** **byte**[toReceivePacket.getLength()];//retrieving actual length of received data

serverData = Arrays.*copyOf*(toReceivePacket.getData(),toReceivePacket.getLength());//copying byte[] response data from buffer of required length

serverMessage = **new** String(serverData,StandardCharsets.*UTF\_8*);//converting byte[] to string message to retrieve checksum

String serverCheckSumString = ClientClass.*getPatternMatch*(2,serverMessage,"^(.+?)(\\d\*)$");//extracting checksum value

**long** serverCheckSum1 = Integer.*parseInt*(serverCheckSumString);//converting it to long

String srvrMsgStrChkSu = ClientClass.*getPatternMatch*(1,serverMessage,"^(.+?)(\\d\*)$");//extracting entire response message except checksum

**byte**[] srvMsgStrChkSuArBy = ClientClass.*convertStringtoByteArray*(srvrMsgStrChkSu);//converting to byte[] for checksum calculation

**long** serverCheckSum2 = ClientClass.*calculateCheckSum*(srvMsgStrChkSuArBy);//calculating checksum for response message(16 bit unsigned integer)

//System.out.println(serverCheckSum2);

//comparing checksum values of response message

**if** (serverCheckSum1 == serverCheckSum2)//comparing calculated and received checksum ,if equality then break & move ahead ,else go to step 2 and repeat sending the request with same measurement id

{**break**;}

**else** {**continue**;}

}//third while loop

//processing as per code value in response

String srvrmsgcodestr = ClientClass.*getPatternMatch*(1,serverMessage,"<code>(.\*?)</code>");//extracting response code

**int** srvrmsgcode = Integer.*parseInt*(srvrmsgcodestr);//extracting it in integer

**switch**(srvrmsgcode){//processing based on response message codes

**case** 0: String srvrmsgmeas = ClientClass.*getPatternMatch*(1,serverMessage,"<value>(.\*?)</value>");System.*out*.println("The measurement value of "+MeasId+" is :" + srvrmsgmeas + "\n");clientSocket.close();**break** outerloop;//extract value if code is 1,break and send a new measurement value again

**case** 1:System.*err*.println("Error: integrity check failure . The request has one or more bit errors");

System.*out*.println("Do you want to send the same measurement request again ?? (y/n)");

**switch**(inp.next())

{**case** "y": **continue**; // deciding course of action based on user input when response code is 1

**case** "n" :System.*exit*(0);} // upon entering "n" program will terminate & upon entering "y", same measurement request is resent

**case** 2: System.*err*.println("Error: malformed request. The syntax of the request message is not correct");System.*exit*(0);

**case** 3: System.*err*.println("Error: non-existant measurement. The measurement with the requested measurement ID does not exist.");System.*exit*(0);

}//switch

}//second while loop

}//first main while loop

}//main method

}//class ClientMain

Server class

package actualProjectPackage;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class ServerClass {

private byte[] bytArray;

private static String msgString;

private static String id;

private static int code;

private static String measurement;

private static String value;

private final long C = 7919;

private final long D = 65536;

private static int groupNo;

private static long measurId;

private static String toMatchStr;

private static String patternTo;

private static String responseMessage;

//constructor method

public ServerClass(byte[] abytArray)

{bytArray = abytArray;

msgString = null;

id = null;

code= 0;

measurement = null;

responseMessage = null;

groupNo = 0;

toMatchStr = null;

patternTo = null;

}

//method to calculate checksum

public long calculateCheckSum(byte[] abytArray)

{bytArray = abytArray;

long S =0;

long index;

if (bytArray.length % 2 == 0)

{long[] unsignedWords = new long[bytArray.length/2];

int k = 0;

for(int i=0;i<bytArray.length;i = i +2)

{long x = (long)(((bytArray[i] & 0xFF) << 8) | (bytArray[i+1] & 0xFF));

unsignedWords[k] = x; k = k +1;}

for (int j = 0; j<unsignedWords.length;j++)//calculating checksum

{index = S ^ unsignedWords[j];

S = ((C\*index) % D);}

return S;}

else //if character sequence is odd in length

{long [] unsignedWords = new long [(bytArray.length+1)/2];

int o = 0;

for (int m=0;m<bytArray.length-1;m = m + 2)//iterating over except last odd element

{long x = (long)(((bytArray[m] & 0xFF) << 8) | (bytArray[m+1] & 0xFF));

unsignedWords[o] = x;o = o + 1;}

unsignedWords[(bytArray.length+1)/2 -1 ] = (long)(((bytArray[bytArray.length-1] & 0xFF) << 8) | (0 & 0xFF));//calculating unsigned word for last odd element

for (int n = 0; n<unsignedWords.length;n++)

{index = S ^ unsignedWords[n];

S = ((C\*index) % D);}

return S;}

}

//method to check syntax of received message which includes checking all opening,closing tags, spelling,index of tags, measurement id & to ensure that it does not contain any invalid character

public static boolean checkSyntax(String aMsgString,long aMeasurId)

{msgString = aMsgString;//add to remove white space here

measurId = aMeasurId;

if (msgString.contains("<request>") && msgString.contains("</request>") && msgString.contains("<id>") && msgString.contains("</id>") && msgString.contains("<measurement>") && msgString.contains("</measurement>") && measurId < (Short.MAX\_VALUE-Short.MIN\_VALUE) && measurId > 0 && measurId == (int)measurId)

{if ((msgString.indexOf("<request>") < msgString.indexOf("<id>")) && (msgString.indexOf("<id>") < msgString.indexOf("</id>")) && (msgString.indexOf("</id>") < msgString.indexOf("<measurement>")) && (msgString.indexOf("<measurement>") < msgString.indexOf("</measurement>")) && (msgString.indexOf("</measurement>") < msgString.indexOf("</request>")))

{if (msgString.matches("^[a-zA-Z0-9<>/]+$"))// to ensure it does not contain any invalid character other than required to create message

{return true;}

else {return false;}

}

else {return false;}

}

else {return false;}

}

//method to create response message

public static String createMessage(String aId,int aCode,String aMeasurement,String aValue)

{id = aId;

code = aCode;

measurement = aMeasurement;

value = aValue;

String toReturn = null;

if (code == 0)

{toReturn = "<response>" + "\n" +"\t"+"<id>" + id +"</id>" + "\n" + "\t" + "<code>" + code +"</code>" +"\n" +"\t"+ "<measurement>" + measurement + "</measurement>" +"\n" + "\t"+"<value>"+ value + "</value>" + "\n" + "</response>";

}

else if (code == 1)

{toReturn = "<response>" + "\n"+"\t" +"<id>" + id +"</id>" + "\n" +"\t"+ "<code>" + code +"</code>" + "\n" + "</response>";}

else if (code ==2){ toReturn = "<response>" + "\n" +"<id>" + id +"</id>" + "\n" + "<code>" + code +"</code>" + "\n" + "</response>";}

else if (code == 3){toReturn = "<response>" + "\n" +"<id>" + id +"</id>" + "\n" + "<code>" + code +"</code>" + "\n" + "</response>";}

return toReturn;

}

//method to convert string object to byte[] for transmission

public static byte[] convertStringtoByteArray(String aResponseMessage)//remove white spaces in main before getting byte[]

{responseMessage = aResponseMessage;

return responseMessage.getBytes();

}

//method to match patterns and extract desired strings using regular expressions

public static String getPatternMatch(int aGroupNo,String aToMatchStr,String aPatternTo)

{

groupNo = aGroupNo;

toMatchStr = aToMatchStr;

patternTo = aPatternTo;

Pattern pat = Pattern.compile(patternTo);

Matcher mat = pat.matcher(toMatchStr);

if (mat.find())

{return mat.group(groupNo);}

else {return "0";}

}

}//class ServerClass

Server main class

package actualProjectPackage;

import java.io.\*;

import java.net.\*;

import java.nio.charset.StandardCharsets;

import java.util.\*;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class ServerMain {

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

// TODO Auto-generated method stub

while(true){

BufferedReader buffRead = null;

String measureId[] = new String[100];

String tempValue[] = new String[100];

//String foundVlaue;

String serverMessage;

String clntMeasIdStr = null;

String servActMsg=null;

String clntId = null;

//long clntMeasId =0;

long servMsgChkSum;

byte[] actServMsgByt = null;

int k =0;int j= 0;

String foundValue = null ;

String line ;

String formattedServMessage;

byte[] servMsgByt;

byte[] serverBuffer = new byte[1000];//initializing buffer with some large size value

DatagramPacket toRecPacket = new DatagramPacket(serverBuffer,serverBuffer.length);//creating receiving packet

DatagramSocket serverSocket = new DatagramSocket(9999);//creating server socket to receive

DatagramSocket toSend = new DatagramSocket(52225);//creating separate socket to send

serverSocket.receive(toRecPacket);//receiving packets

int clientPort = toRecPacket.getPort();//extracting client's port number

InetAddress clientAddress = toRecPacket.getAddress();//extracting client's address

String origClntMsg = null;

byte[] clientDataArray = new byte[toRecPacket.getLength()];//creating a byte[] of received data length

clientDataArray = Arrays.copyOf(toRecPacket.getData(),toRecPacket.getLength());

int i =0;

int index = 0;

int[] ind = new int[6];//creating array to store repetitions of char > to segregate checksum value

char search = '>';//character to search

byte toSearch = (byte)search;//getting ASCII value of > to search for

for(int g=0;g<clientDataArray.length;g++)

{if(clientDataArray[g] == toSearch)//creating an array of indices occurrences of char > to pick maximum

{ind[i] = g;

i++;

}

}

for (int t=0;t<ind.length;t++)

{if (ind[t] > index){index = ind[t];} //getting the highest index for occurrence of >

}

byte[] clientChkSumByte = Arrays.copyOfRange(clientDataArray,index+1,clientDataArray.length);//extracting byte[] of check sum

String chkSumStr = new String(clientChkSumByte,StandardCharsets.UTF\_8);//converting byte[] of checksum to string

long chkSumVal = Integer.parseInt(chkSumStr);//extracting checksum value sent by client from string as unsigned integer

byte[] clientMessageByte = Arrays.copyOfRange(clientDataArray,0,index+1);//extracting byte[] of message excluding checksum

ServerClass aObject = new ServerClass(clientMessageByte);//creating object

long msgChkSum = aObject.calculateCheckSum(clientMessageByte);//calculating checksum of message from byte[]

if (msgChkSum == chkSumVal)// if condition to check for integrity checksum

{origClntMsg = new String(clientDataArray,StandardCharsets.UTF\_8);//extracting entire message string if it passes integrity checksum

clntId = ServerClass.getPatternMatch(1,origClntMsg,"<id>(.\*?)</id>");

clntMeasIdStr = ServerClass.getPatternMatch(1,origClntMsg,"<measurement>(.\*?)</measurement>");//extracting measurement id

long clntMeasdIdInt = Integer.parseInt(clntMeasIdStr);

//System.out.print(origClntMsg);

if (ServerClass.checkSyntax(origClntMsg,clntMeasdIdInt)) // if condition to check for syntax i.e. all tags,placement and to check for 16 bit unsigned integer i.e. measurement id according to step 3

{

try{

buffRead = new BufferedReader(new FileReader("C:\\Users\\comp\\Downloads\\data.txt"));} //reading value corresponding to measurement id

catch (FileNotFoundException e) {System.out.println("File not found");}

while ((line = buffRead.readLine()) != null)

{String info[] = line.split("\\s+");

measureId[k] = info[0];

tempValue[j] = info[1];

k++;

j++;}

//System.out.println("\n"+clntMeasIdStr);

if (Arrays.asList(measureId).contains(clntMeasIdStr))// third if condition to check for valid measurement id

{for (int y=0;y<measureId.length;y++)

{if (measureId[y].equals(clntMeasIdStr))//

{foundValue = tempValue[y];}}

serverMessage = ServerClass.createMessage(clntId ,0,clntMeasIdStr,foundValue);//creating message when code is 0

servMsgByt = ServerClass.convertStringtoByteArray(serverMessage.replaceAll("\\s+",""));//converting message to byte array

ServerClass aObj = new ServerClass(servMsgByt);

servMsgChkSum = aObj.calculateCheckSum(servMsgByt);//calculating checksum for created response message

servActMsg = serverMessage + servMsgChkSum;//appending check sum to response message

formattedServMessage = servActMsg.replaceAll("\\s+","");//removing all white spaces before sending out

System.out.print("\n"+servActMsg+"\n");

actServMsgByt = ServerClass.convertStringtoByteArray(formattedServMessage);//converting entire response message including checksum to byte[]

DatagramPacket toSendPacket = new DatagramPacket(actServMsgByt,actServMsgByt.length,clientAddress,clientPort);//creating datagrampacket to send

toSend.send(toSendPacket);serverSocket.close();toSend.close();

}

else //sending message when measurement id does not match i.e. code 3

{serverMessage = ServerClass.createMessage(clntId,3,null,null);

servMsgByt = ServerClass.convertStringtoByteArray(serverMessage.replaceAll("\\s+",""));//converting message to byte array

ServerClass aObj = new ServerClass(servMsgByt);

servMsgChkSum = aObj.calculateCheckSum(servMsgByt);//calculating checksum for created response message

servActMsg = serverMessage + servMsgChkSum;//appending check sum to response message

formattedServMessage = servActMsg.replaceAll("\\s+","");//removing all white spaces before sending out

System.out.print("\n"+servActMsg + "\n");

actServMsgByt = ServerClass.convertStringtoByteArray(formattedServMessage);//converting entire response message including checksum to byte[]

DatagramPacket toSendPacket = new DatagramPacket(actServMsgByt,actServMsgByt.length,clientAddress,clientPort);//creating datagrampacket to send

toSend.send(toSendPacket);serverSocket.close();toSend.close();

}

}

else //sending message when syntax does not match i.e. code 2

{clntId = ServerClass.getPatternMatch(1,origClntMsg,"<id>(.\*?)</id>");

serverMessage = ServerClass.createMessage(clntId,2,null,null);

servMsgByt = ServerClass.convertStringtoByteArray(serverMessage.replaceAll("\\s",""));//converting message to byte array

ServerClass aObj = new ServerClass(servMsgByt);

servMsgChkSum = aObj.calculateCheckSum(servMsgByt);//calculating checksum for created response message

servActMsg = serverMessage + servMsgChkSum;//appending check sum to response message

formattedServMessage = servActMsg.replaceAll("\\s+","");//removing all white spaces before sending out

System.out.print("\n"+servActMsg + "\n");

actServMsgByt = ServerClass.convertStringtoByteArray(formattedServMessage);//converting entire response message including checksum to byte[]

DatagramPacket toSendPacket = new DatagramPacket(actServMsgByt,actServMsgByt.length,clientAddress,clientPort);//creating datagrampacket to send

toSend.send(toSendPacket);serverSocket.close();toSend.close();}

}

else //sending message when integrity checksum fails i.e. code 1

{serverMessage = ServerClass.createMessage(null,1,null,null);

servMsgByt = ServerClass.convertStringtoByteArray(serverMessage.replaceAll("\\s",""));//converting message to byte array

ServerClass aObj = new ServerClass(servMsgByt);

servMsgChkSum = aObj.calculateCheckSum(servMsgByt);//calculating checksum for created response message

servActMsg = serverMessage + servMsgChkSum;//appending check sum to response message

//System.out.println(servMsgChkSum);

formattedServMessage = servActMsg.replaceAll("\\s+","");//removing all white spaces before sending out

System.out.print("\n"+servActMsg+"\n");

actServMsgByt = ServerClass.convertStringtoByteArray(formattedServMessage);//converting entire response message including checksum to byte[]

DatagramPacket toSendPacket = new DatagramPacket(actServMsgByt,actServMsgByt.length,clientAddress,clientPort);//creating datagrampacket to send

toSend.send(toSendPacket);serverSocket.close();toSend.close();

}//else condition

}//while condition

}//main method

}//class ServerMain