/\*  
 1. Name / Date: satya Yoganand Addala / 05-11-2022  
  
 2. Java version used (java -version), if not the official version for the class: 18.0.2  
  
 3. Precise command-line compilation examples / instructions:  
 > javac AsyncJokeServer.java  
 > javac AsyncJokeClientAdmin.java  
 > javac AsyncJokeClient.java  
  
 4. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java AsyncJokeServer.java  
 > java AsyncJokeClient.java  
 > java AsyncJokeClientAdmin.java  
  
 All acceptable commands are displayed on the various consoles.  
  
 5. Notes:  
  
 \* Implemented Async Joke Server gets started with the udp connection on the port 9999 and for Async Joke client Admin I am using tcp with the port 5050  
 \* Implemented an Addition Looper to start processing the addition requests till the joke/proverb response appears.  
 \* There is a Unique ID used for each client and based on this i have used HapMap which holds Unique Id and set of Jokes  
 \* There is a Joke Indicator Included which checks if the jokes in the list exceeds 4 if it does them it deletes all the four jokes.  
 \* There is a Proverb Indicator Included which checks if the jokes in the list exceeds 4 if it does them it deletes all the four Proverbs.  
 \* In this I have used random method to shuffle between jokes and a condition to remove duplicates.  
 \* Methods RemoveJokes() and RemoveProverbs() are used for removing jokes/proverbs respectively from the list.  
 \* Jokes have been taken from https://www.rd.com/list/funniest-one-liners-you-havent-heard-yet/  
 \* I have used a piece of code that I got referenced from web for converting the response bytes to string and here is the link: https://www.geeksforgeeks.org/working-udp-datagramsockets-java/  
 \* Modified the previously implemented JokeServer Code to accept the UDP Connections.  
  
\*/  
  
import java.io.IOException;  
import java.net.\*;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintStream;  
import java.util.\*;  
import java.util.concurrent.TimeUnit;  
  
public class AsyncJokeServer {  
 /\* Initializing the mode to Joke by default \*/  
 public static String *mode* = "joke";  
 /\* Setting the serverType to primary by default \*/  
 public static boolean *serverType* = false;  
  
 public static void main(String args[]) throws IOException, InterruptedException {  
 /\* Initializing servsock to null \*/  
 DatagramSocket ds = new DatagramSocket(9999);  
  
 if (args.length < 1) { /\* Checking for Arguments \*/  
 System.*out*.println("Satya Yoganand's Joke server starting up...,listening at port 9999.\n");  
 }  
  
 /\* Creating an Admin Looper Thread \*/  
 AdminLooper AL = new AdminLooper();  
 Thread t = new Thread(AL);  
 /\* Starting the thread it, waiting for administration input \*/  
 t.start();  
  
 Worker worker = new Worker(ds);  
 Thread w = new Thread(worker);  
  
 w.start();  
 }  
  
 /\* Method for converting received byte data to string \*/  
 public static StringBuilder data(byte[] b)  
 {  
 /\* Checking null for byte \*/  
 if (b == null)  
 /\* Return Null \*/  
 return null;  
 /\* Forming String builder \*/  
 StringBuilder result = new StringBuilder();  
 /\* Initializing i \*/  
 int i = 0;  
 /\* Looping till byte is 0 \*/  
 while (b[i] != 0)  
 {  
 /\* Appending result to the String Builder \*/  
 result.append((char) b[i]);  
 /\* Incrementing i \*/  
 i++;  
 }  
 return result;  
 }  
 }  
  
class AdminLooper implements Runnable {  
 public static boolean *adminControlSwitch* = true;  
 public void run() { /\* Running the Admin listen loop \*/  
 /\* Number of requests for OS to queue \*/  
 int q\_len = 6;  
 int port;  
  
 port = 5050;  
 System.*out*.println("Satya Yoganand's Admin Looper starting up...,listening at port : "+port +".\n");  
 /\* Creating Socket Variable \*/  
 Socket sock;  
 ServerSocket servsock = null;  
 try {  
 servsock = new ServerSocket(port, q\_len);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 while (*adminControlSwitch*) {  
 /\* Accepting the connections \*/  
 try {  
 sock = servsock.accept();  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 /\* Starting the ModeWorker \*/  
 new ModeWorker(sock).start();  
 }  
 }  
}  
  
class ModeWorker extends Thread {  
  
 /\* Creating a Socket Variable \*/  
 Socket sock;  
 // Constructor for fetching the Socket  
 ModeWorker(Socket s) {  
 sock = s;  
 }  
 public void run() {  
 /\* Initializing out variable to null and this is used to send the data from Server to Client \*/  
 PrintStream out = null;  
 /\* Initializing in variable to null and this is used to receive data from client to server \*/  
 BufferedReader in = null;  
  
 try {  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Used to send information from the server to client through PrintStream \*/  
 out = new PrintStream(sock.getOutputStream());  
  
 try {  
 String mode;  
 /\* Blocking Call - the server code will pauses at this point and starts listening to the input from client \*/  
 /\* This will capture the mode if it is a Joke or Proverb \*/  
 mode = in.readLine();  
  
 if(!AsyncJokeServer.*serverType*) {System.*out*.println("Switching Mode to : " + mode );}  
 else{System.*out*.println("<S2> Switching Mode to : " + mode );}  
 if(mode.equalsIgnoreCase("joke")) {  
 /\* Setting the mode to Joke \*/  
 AsyncJokeServer.*mode* = "Joke";  
 /\* Checking the serverType \*/  
 if(!AsyncJokeServer.*serverType*){System.*out*.println("Mode set to Joke ");}  
 else{System.*out*.println("<S2> Mode set to Joke ");}  
 }  
 if(mode.equalsIgnoreCase("proverb")){  
 /\* Setting the mode to Proverb \*/  
 AsyncJokeServer.*mode* = "Proverb";  
 /\* Checking the server Type \*/  
 if(!AsyncJokeServer.*serverType*) {System.*out*.println("Mode set to Proverb");}  
 else {System.*out*.println("<S2> Mode set to Proverb");}  
 }  
 else {  
 /\* Checking the serverType \*/  
 if(!AsyncJokeServer.*serverType*){System.*out*.println("Bad Request");}  
 else {System.*out*.println("<S2> Bad Request");}  
 }  
 } catch (IOException x) { /\* Handling the IOException if the name variable is not valid \*/  
 if(!AsyncJokeServer.*serverType*) {System.*out*.println("Server read error");}  
 else{System.*out*.println("<S2> Server read error");}  
 x.printStackTrace();  
 }  
 sock.close();  
  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
  
 }  
}  
class Worker extends Thread {  
  
 //Create a Socket Object  
 public static Socket *sock*;  
 /\* Array List to contain all the Jokes \*/  
 private static ArrayList<String> *Jokeslist* = new ArrayList<String>();  
  
 /\* Array List to contain all the proverbs \*/  
 private static ArrayList<String> *Proverbslist* = new ArrayList<String>();  
  
 /\* Array List to contain all the copies of Jokes that are used \*/  
 private static ArrayList<String> *JokeslistCopy* = new ArrayList<String>();  
  
 /\* Hashmap to map main the conversations of client and server with respect to userId\*/  
 private static HashMap<String, ArrayList<String>> *map* = new HashMap<>();  
  
 /\* Array List to contain all the copies of Proverbs that are used \*/  
 private static ArrayList<String> *ProverbslistCopy* = new ArrayList<String>();  
  
 /\* List for holding all the unique userID's\*/  
 private static LinkedList<String> *UUIDList* = new LinkedList<>();  
  
 /\* Creating a static variable for username \*/  
 private static String *user\_Name*;  
  
 /\* This Joke\_Indicator will help to keep track of the Jokes completed per cycle\*/  
 private static int *Joke\_Indicator* = 0;  
  
 /\* This Proverb\_Indicator will help to keep track of the Procerb completed per cycle\*/  
 private static int *Proverb\_Indicator* = 0;  
  
 /\* Initializing the out variable to null \*/  
 private static PrintStream *out* = null;  
  
 /\* Initializing the in variable to null \*/  
 private static BufferedReader *in* = null;  
  
 /\* Create a Datagram Socket \*/  
 DatagramSocket socket;  
  
 /\* Constructor for fetching the Socket \*/  
  
 public Worker(DatagramSocket ds){  
 this.socket = ds;  
 }  
  
 public void run() {  
 /\* Accepting and sending all the Sockets from Client \*/  
 while(true){  
  
 try {  
  
 byte[] receive = new byte[65535];  
 /\* Creating a datagram packet object and initializing it to null \*/  
 DatagramPacket DpReceive = null;  
 /\* Fetching the Inet Address \*/  
 InetAddress ip = InetAddress.*getLocalHost*();  
  
 try {  
 String name;  
 String userId;  
 int port = 0;  
 String address = "";  
 /\* Initializing the new Datagram Packet to receive packets from clients \*/  
 DpReceive = new DatagramPacket(receive, receive.length);  
 /\* Fetching the packet \*/  
 socket.receive(DpReceive);  
  
 /\* Storing User ID \*/  
 userId = AsyncJokeServer.*data*(receive).toString().split("@")[0];  
 /\* Storing Name \*/  
 name = AsyncJokeServer.*data*(receive).toString().split("#")[0].split("@")[1];  
 /\* Storing UDP Port Number \*/  
 port = Integer.*parseInt*(AsyncJokeServer.*data*(receive).toString().split("#")[1].split(";")[0]);  
 /\* Storing InetAddress \*/  
 address = AsyncJokeServer.*data*(receive).toString().split(";")[1];  
  
 /\* Printing the obtained details \*/  
 System.*out*.println("UserID,Name : "+ userId + " "+name);  
 System.*out*.println("Port Number : "+port +", Address :"+address);  
  
 if(!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 System.*out*.println("UUID for the current Session : " + userId);  
 System.*out*.println("Providing a "+ AsyncJokeServer.*mode* +" ......");  
 }  
 else{  
 System.*out*.println("<S2> UUID for the current Session : " + userId);  
 System.*out*.println("<S2> Providing a "+ AsyncJokeServer.*mode* +" ......");  
 }  
 /\* Initiating Addition Looper Thread \*/  
 AdditionLooper Al = new AdditionLooper(port,address);  
 Thread t = new Thread(Al);  
 /\* Starting the thread \*/  
 t.start();  
 /\* Inducing sleep for 40 seconds \*/  
 TimeUnit.*SECONDS*.sleep(40);  
 /\* Invoking RandomSelection method for generating random Jokes/Proverbs \*/  
 *randomSelection*(userId,name,DpReceive,socket);  
  
 } catch (IOException x) { /\* Handling the IOException if the name variable is not valid \*/  
 if(!AsyncJokeServer.*serverType*){System.*out*.println("Server read error");}  
 else{System.*out*.println("<S2> Server read error");}  
 x.printStackTrace();  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 } catch (IOException ioe) {  
 System.*out*.println(ioe);  
 }}  
 }  
  
 static void randomSelection(String userId,String name,DatagramPacket dp,DatagramSocket ds) throws IOException {  
 *user\_Name* = name;  
 String Random\_Joke = null;  
 String jokeCycle\_Status = null;  
  
 String Random\_Proverb = null;  
 String proverbCycleStatus = null;  
 /\* Checking for the userId in UUID List \*/  
 if (*UUIDList*.contains(userId)) {  
 if (!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 System.*out*.println("UserId Already Exists !!!!");  
 } else {  
 System.*out*.println("<S2> UserId Already Exists !!!!");  
 }  
  
 if (AsyncJokeServer.*mode*.equalsIgnoreCase("joke")) {  
 String random\_Joke = *randomJoke*();  
 /\* Loop for preventing Duplicate Jokes from adding to Hashmap \*/  
 while (*map*.get(userId).contains(random\_Joke)) {  
 random\_Joke = *randomJoke*();  
 }  
 /\* Adding the Joke to Hashmap for respective UserId \*/  
 *map*.get(userId).add(random\_Joke);  
 int jokesCopyListSize = *JokeslistCopy*.size();  
  
 /\* Checking for Elements in Hashmap and if they are greater than or equal to 4 then incrementing the Joke\_Indicator and if it is  
 \* equal to 4 then remove the four jokes from hashmap and reset the counter \*/  
 if (*map*.get(userId).stream().count() >= 4) {  
 /\* Resetting the joke Indicator after every Cycle \*/  
 *Joke\_Indicator* = 0;  
 for (String joke : *map*.get(userId).stream().toList()) {  
 if (joke.startsWith("J")) {  
 /\* Incrementing Joke Indicator \*/  
 *Joke\_Indicator*++;  
 if (*Joke\_Indicator* == 4) { /\* Checking for four Jokes in the Hashmap arraylist\*/  
 /\* Calling RemoveJokes to get the remove count of Jokes \*/  
 int rmCount = *RemoveJokes*(*map*, userId);  
 /\* Resetting the Joke Indication with the number of jokes removed \*/  
 *Joke\_Indicator* = *Joke\_Indicator* - rmCount;  
  
 }  
 }  
 }  
 }  
  
 if (!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 /\* Storing random Joke to Client \*/  
  
 Random\_Joke = random\_Joke;  
  
 if (jokesCopyListSize == 0) {  
 jokeCycle\_Status = "Joke Cycle Completed";  
 }  
 }  
 }  
 /\* Server mode is set to Proverb \*/  
 else if (AsyncJokeServer.*mode*.equalsIgnoreCase("proverb")) {  
 String random\_Proverb = *randomProverb*();  
 /\* Loop for preventing duplicate proverbs from adding to Hashmap \*/  
 while (*map*.get(userId).contains(random\_Proverb)) {  
 random\_Proverb = *randomProverb*();  
 }  
 /\* Adding the Proverb to Hashmap with respect to the UserId\*/  
 *map*.get(userId).add(random\_Proverb);  
 int proverbCopyListSize = *ProverbslistCopy*.size();  
 /\* Checking for Elements in Hashmap and if they are greater than or equal to 4 then incrementing the Proverb\_Indicator and if it is  
 \* equal to 4 then remove the four Proverbs from hashmap and reset the counter \*/  
 if (*map*.get(userId).stream().count() >= 4) {  
 /\* Resetting the joke Indicator after every Cycle \*/  
 *Proverb\_Indicator* = 0;  
 for (String proverb : *map*.get(userId).stream().toList()) {  
 if (proverb.startsWith("P")) {  
 /\* Incrementing Proverb Indicator \*/  
 *Proverb\_Indicator*++;  
 if (*Proverb\_Indicator* == 4) { /\* checking for 4 proverbs in the arraylist of the hashmap \*/  
 /\* Calling Remove Proverbs to get the remove count of proverbs \*/  
 int rmCount = *RemoveProverbs*(*map*, userId);  
 /\* Resetting the Joke Indication with the number of jokes removed \*/  
 *Proverb\_Indicator* = *Proverb\_Indicator* - rmCount;  
 }  
 }  
 }  
 }  
 if (!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 /\* Storing random proverb to Client \*/  
 Random\_Proverb = random\_Proverb;  
 if (proverbCopyListSize == 0) {  
 proverbCycleStatus = "Proverb Cycle Completed ...";  
 }  
 }  
 }  
 } else {  
 if (!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 System.*out*.println("Creating new User...");  
 } else {  
 System.*out*.println("<S2> Creating new User...");  
 }  
 /\* Adding the Userid to UUIDList \*/  
 *UUIDList*.add(userId);  
 if (AsyncJokeServer.*mode*.equalsIgnoreCase("joke")) {  
 String random\_Joke = *randomJoke*();  
  
 /\* Creating a seperate ArrayList for the newly created userId and adding the first joke to Hashmap with the userId \*/  
 *map*.put(userId, new ArrayList<>());  
 *map*.get(userId).add(random\_Joke);  
  
 if (!AsyncJokeServer.*serverType*) { /\* Checking Server Type \*/  
 Random\_Joke = random\_Joke;  
 }  
 } else if (AsyncJokeServer.*mode*.equalsIgnoreCase("proverb")) {  
 String random\_Proverb = *randomProverb*();  
  
 /\* Creating a seperate ArrayList for the newly created userId and adding the first proverb to Hashmap with the userId \*/  
 *map*.put(userId, new ArrayList<>());  
 *map*.get(userId).add(random\_Proverb);  
  
 if (!AsyncJokeServer.*serverType*) {/\* Checking Server Type \*/  
 /\* printing random proverb in primary server \*/  
 Random\_Proverb = random\_Proverb;  
 }  
 }  
 }  
 /\* Creating a byte variable to send the data \*/  
 byte[] buf = null;  
 /\* Concatinating the data that has to be sent \*/  
 String packetData = Random\_Joke +"joke"+ Random\_Proverb+ "proverb" + proverbCycleStatus +"cycleStatus"+ jokeCycle\_Status;  
 /\* Converting it to bytes \*/  
 buf = packetData.getBytes();  
 /\* Creating a datagram packet to send the joke/proverb/cycle status to the client \*/  
 DatagramPacket dp3 = new DatagramPacket(buf, buf.length, dp.getAddress(), dp.getPort());  
 /\* Sending the data \*/  
 ds.send(dp3);  
  
 }  
  
 /\* Method for removing Jokes when Joke cycle Completes \*/  
 static int RemoveJokes(Map<String,ArrayList<String>> map,String userId){  
 int Remove\_Count = 0;  
 for(String joke : map.get(userId).stream().toList()) {  
 if (joke.contains("J")) { /\* Removing Jokes from hashmap for the userId \*/  
 map.get(userId).remove(joke);  
 /\* Incrementing remove count \*/  
 Remove\_Count++;  
 }  
 }  
 /\* Returning the count of removed jokes \*/  
 return Remove\_Count;  
 }  
  
 /\* Method for removing Proverbs when Proverb cycle Completes \*/  
 static int RemoveProverbs(Map<String,ArrayList<String>> map,String userId){  
 int Remove\_Count = 0;  
 for(String joke : map.get(userId).stream().toList()) {  
 if (joke.contains("P")) { /\* Removing Proverbs from hashmap for the userId \*/  
 map.get(userId).remove(joke);  
 /\* Incrementing remove count \*/  
 Remove\_Count++;  
 }  
 }  
 /\* Returning the count of removed proverbs \*/  
 return Remove\_Count;  
 }  
  
 /\* Method for fetching random Joke from the Jokes List \*/  
 static String randomJoke() throws IOException {  
  
 // out = new PrintStream(sock.getOutputStream());  
 /\* Clearing the JokesList \*/  
 *Jokeslist*.clear();  
 *Jokeslist*.add("JA "+*user\_Name* + ":"+" What did one DNA say to the other DNA? “Do these genes make me look fat?”");  
 *Jokeslist*.add("JB "+*user\_Name*+ ":" +" My IQ test results came back. They were negative.");  
 *Jokeslist*.add("JC "+*user\_Name*+ ":" +" What do you get when you cross a polar bear with a seal? A polar bear.");  
 *Jokeslist*.add("JD "+*user\_Name*+ ":" +" Why was six afraid of seven? Because seven eight nine.");  
 /\* Removing Joke Duplicates \*/  
 *removeJokeDuplicates*();  
 /\* Initializing the random method \*/  
 Random random = new Random();  
 int select = random.nextInt(*Jokeslist*.size());  
 String selectedJoke = *Jokeslist*.get(select);  
 /\* Adding selectedjoke to the jokelistCopy \*/  
 *JokeslistCopy*.add(selectedJoke);  
 /\* Calculating the size of JokeCopylist \*/  
 int jokesCopyListSize = *JokeslistCopy*.size();  
 if (jokesCopyListSize==4) {  
 /\* Clearing the JokeCopyList \*/  
 *JokeslistCopy*.clear();  
 if(!AsyncJokeServer.*serverType*) {  
 System.*out*.println("Joke Cycle has been Completed ...");  
 }  
 else {System.*out*.println("<S2> Joke Cycle has been Completed ...");  
 }  
 }  
 /\* Returning the selected Joke \*/  
 return selectedJoke;  
 }  
  
 /\* Method for fetching random proverb from Proverbs List \*/  
 static String randomProverb(){  
 /\*Clearing the proverb List \*/  
 *Proverbslist*.clear();  
 *Proverbslist*.add("PA "+*user\_Name* +":"+" Honesty is the best policy.");  
 *Proverbslist*.add("PB "+*user\_Name* +":"+ " Strike while the iron is hot.");  
 *Proverbslist*.add("PC "+*user\_Name* +":"+" Don’t judge a book by its cover.");  
 *Proverbslist*.add("PD "+*user\_Name* +":"+" An apple a day keeps the doctor away.");  
 /\* Removing the proverb Duplicates \*/  
 *removeProverbDuplicates*();  
 /\* Initializing the random method \*/  
 Random random = new Random();  
 /\* Getting the size of proverbslist \*/  
 int select = random.nextInt(*Proverbslist*.size());  
 String selectedProverb = *Proverbslist*.get(select);  
 /\* Adding the selected proverb to proverbs copy list \*/  
 *ProverbslistCopy*.add(selectedProverb);  
 /\* Calculating the size of proverb copy list \*/  
 int proverbCopyListSize = *ProverbslistCopy*.size();  
 if (proverbCopyListSize==4) {  
 /\* Clearing the proverCopylist \*/  
 *ProverbslistCopy*.clear();  
 if(!AsyncJokeServer.*serverType*) {System.*out*.println("Proverb Cycle has been Completed ...");}  
 else{System.*out*.println("<S2> Proverb Cycle has been Completed ...");}  
 }  
 /\* Returning the selected proverb \*/  
 return selectedProverb;  
 }  
  
 /\* Method for removing Joke Duplicates from Copied JokeList \*/  
 static void removeJokeDuplicates(){  
 for(String joke : *JokeslistCopy*){  
 if(*Jokeslist*.contains(joke)){  
 *Jokeslist*.remove(joke);  
 }  
 }  
 }  
  
 /\* Method for removing Proverb Duplicates from Copied ProverbList \*/  
 static void removeProverbDuplicates(){  
 for(String proverb : *ProverbslistCopy*){  
 if(*Proverbslist*.contains(proverb)){  
 *Proverbslist*.remove(proverb);  
 }  
  
 }  
 }  
  
}  
/\* Addition looper thread to perform sum of the two digits \*/  
class AdditionLooper extends Thread {  
 /\* Initializing a datagram Socket \*/  
 DatagramSocket socket;  
 int port;  
 String address;  
 int num1 =0;  
 int num2 =0;  
 int result = 0;  
 String result\_String = "";  
 byte[] results;  
 /\* Constructor to receive port and Inet Address \*/  
 AdditionLooper(int port,String address){  
 this.port = port;  
 this.address = address;  
 }  
 public void run() {  
  
 try {  
 /\* Checking if the datagram socket is null \*/  
 if(socket == null) {  
 socket = new DatagramSocket(port);  
 }  
 System.*out*.println("Started Addition looper");  
  
 /\* Accepting and sending all the packets \*/  
  
 while(true) {  
  
 try {  
 byte[] buf = new byte[1024];  
 /\* Datagram packet to receive the data from client \*/  
 DatagramPacket dp = new DatagramPacket(buf,buf.length);  
 /\* Receiving data \*/  
 socket.receive(dp);  
 /\* Storing the received data in a string \*/  
 String receivedData = AsyncJokeServer.*data*(buf).toString();  
 /\* Printing the received data \*/  
 System.*out*.println("Printing received Digits : " + receivedData);  
 /\* Fetching the digits from the received data \*/  
 num1 = Integer.*parseInt*(receivedData.split(" ")[0].trim());  
 num2 = Integer.*parseInt*(receivedData.split(" ")[1].trim());  
 /\* Performing Addition \*/  
 result = num1 + num2;  
 /\* Converting the result to string \*/  
 result\_String = result + "";  
 /\* Converting the string to bytes \*/  
 results = result\_String.getBytes();  
 /\* Datagram packet to send the data to client \*/  
 DatagramPacket dp1 = new DatagramPacket(results, results.length, dp.getAddress(), dp.getPort());  
 /\* Sending the data to client \*/  
 socket.send(dp1);  
  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 catch (SocketException e) {  
 }  
 }  
}

/\*  
\* 1. Name / Date: satya Yoganand Addala / 05-11-2022  
  
 2. Java version used (java -version), if not the official version for the class: 18.0.2  
  
 3. Precise command-line compilation examples / instructions:  
 > javac AsyncJokeServer.java  
 > javac AsyncJokeClientAdmin.java  
 > javac AsyncJokeClient.java  
  
 4. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java AsyncJokeServer.java  
 > java AsyncJokeClient.java  
 > java AsyncJokeClientAdmin.java  
  
 All acceptable commands are displayed on the various consoles.  
  
 5. Notes:  
  
 \* Implemented UDP for making a conversation, and I have used port 9999 for this connection.  
 \* Also, I have implemented a free port for the addition conversation to happen  
 \* Induced a sleep of 40 sec in between the responses  
 \* By Executing this file the client will ask the server for a joke/proverb and in between receiving of messages the client will start the addition looper to fetch the result of the entered values.  
 \* Generating Unique ID through UUID.randomUUID() and storing it in Server.  
 \* At the end of the joke/proverb a cycle completed message will get displayed  
 \* Modified the previously implemented JokeClient to accept UDP Connections.  
 \* I have taken reference for the next free port from stackoverflow and here is the link : https://stackoverflow.com/questions/2675362/how-to-find-an-available-port  
 \* I have used a piece of code that i got referenced from web for converting the response bytes to string and here is the link: https://www.geeksforgeeks.org/working-udp-datagramsockets-java/  
  
\*/  
  
import java.io.\*;  
import java.net.\*;  
import java.util.Scanner;  
import java.util.UUID;  
import java.util.concurrent.TimeUnit;  
  
public class AsyncJokeClient {  
  
 public static Boolean *dataRec* = false;  
 public static Boolean *sumFlag*;  
  
 public static void main(String args[]) throws IOException {  
 /\* Server used is localhost \*/  
 String serverName;  
 boolean serverType = false;  
 /\* Creating port variable \*/  
 int port1;  
 if (args.length < 1) {  
 /\* Setting the Severname to localhost \*/  
 serverName = "localhost";  
 System.*out*.println("Satya Yoganand's Joke Client, 1.8.\n");  
 System.*out*.println("Using Server : " + serverName + ", Port: 9999");  
 }  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 ServerSocket s = new ServerSocket(0);  
 /\* Initializing port to a next available port for the Addition thread udp connections \*/  
 port1 = s.getLocalPort();  
 if (!serverType) { /\* Checking ServerType \*/  
 System.*out*.println("Enter the User Name, (quit) to end: ");  
 }  
 System.*out*.flush();  
 String name;  
 name = in.readLine();  
 while (name.equals("")) {  
  
 System.*out*.println("Please enter a valid name : ");  
 name = in.readLine();  
 }  
 try {  
 String nextLine;  
 /\* Generating a random UUID for each client \*/  
 String userId = UUID.*randomUUID*().toString();  
  
 do {  
 System.*out*.println("Enter the User Name or Enter, (quit) to end:");  
 /\* Entering the Domain name or IP Address \*/  
 nextLine = in.readLine();  
 /\*If Entered quit then the process gets stopped or else it will Display jokes /proverbs as requested \*/  
 if (nextLine.indexOf("quit") < 0)  
 *displayJokes*(userId, name,port1);  
 } while (nextLine.indexOf("quit") < 0);  
 System.*out*.println("Cancelled by user request.");  
 } catch (IOException x) { /\* Catch for handling the IOException\*/  
 x.printStackTrace();  
 }  
 }  
  
 static void displayJokes(String userId, String name,int portp1) {  
  
 /\* Variable for receiving Joke,Proverb,Joke Cycle Status and Proverb Cycle Status from server \*/  
  
 String random\_Joke;  
 String random\_Proverb;  
 String proverb\_Cycle\_Status = "";  
 String joke\_Cycle\_Status = "";  
  
 try {  
  
 /\* Creating a datagram socket to establish udp connections \*/  
 DatagramSocket ds = new DatagramSocket();  
 /\* Fetching the InetAddress \*/  
 InetAddress ip = InetAddress.*getLocalHost*();  
 byte buf[] = null;  
 /\* Concatinating the userID,Name,Port number and address \*/  
 String inp = userId + "@" + name + "#" + portp1+";"+ip;  
 /\* Converting it to bytes \*/  
 buf = inp.getBytes();  
 /\* Creating a datagram packet to send the data to server \*/  
 DatagramPacket DpSend = new DatagramPacket(buf, buf.length, ip, 9999);  
 /\* sending data to server \*/  
 ds.send(DpSend);  
  
 AsyncJokeClient.*sumFlag* = true;  
 /\* Initializing the addition looper thread \*/  
 Addition\_looper al = new Addition\_looper(portp1,Thread.*currentThread*());  
 /\* Creating a thread object \*/  
 Thread t = new Thread(al);  
 /\* Starting the thread \*/  
 t.start();  
  
 byte[] b1 = new byte[1024];  
 /\* Datagram Packet to receive the data from server \*/  
 DatagramPacket dp1 = new DatagramPacket(b1, b1.length);  
 /\* Receiving data from server \*/  
 ds.receive(dp1);  
  
 AsyncJokeClient.*sumFlag* = false;  
 /\* Fetching the joke/proverb/cyclestatus and storing them in variables \*/  
 random\_Joke = *data*(b1).toString().split("joke")[0].toString();  
 random\_Proverb = *data*(b1).toString().split("joke")[1].split("proverb")[0].toString();  
 joke\_Cycle\_Status = *data*(b1).toString().split("proverb")[1].split("cycleStatus")[1].toString();  
 proverb\_Cycle\_Status = *data*(b1).toString().split("proverb")[1].split("cycleStatus")[0].toString();  
 /\* Making the thread to wait till the addition is completed \*/  
 AsyncJokeClient.*dataRec* = false;  
 /\* Waiting the current thread till the Addition looper completes \*/  
 t.join();  
 /\* Printing the received data on the server \*/  
 if(!(random\_Joke.equals("null"))){  
 System.*out*.println("Joke Received");  
 System.*out*.println(random\_Joke);  
 }  
 if(!(random\_Proverb.equals("null"))){  
 System.*out*.println("Proverb Received");  
 System.*out*.println(random\_Proverb);  
 }  
 if(!(joke\_Cycle\_Status.equals("null"))){  
  
 System.*out*.println(joke\_Cycle\_Status);  
 }  
 if(!(proverb\_Cycle\_Status.equals("null"))){  
  
 System.*out*.println(proverb\_Cycle\_Status);  
 }  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
 public static StringBuilder data(byte[] b)  
 {  
 /\* Checking null for byte \*/  
 if (b == null)  
 /\* Return Null \*/  
 return null;  
 /\* Forming String builder \*/  
 StringBuilder result = new StringBuilder();  
 /\* Initializing i \*/  
 int i = 0;  
 /\* Looping till byte is 0 \*/  
 while (b[i] != 0)  
 {  
 /\* Appending result to the String Builder \*/  
 result.append((char) b[i]);  
 /\* Incrementing i \*/  
 i++;  
 }  
 return result;  
 }  
}  
/\* Addition looper thread to perform additions \*/  
class Addition\_looper extends Thread{  
 /\* Creating datagramSock object \*/  
 DatagramSocket socket;  
 int port;  
 byte[] buf;  
 byte[] receive;  
 String result;  
 Thread t;  
 /\* Initializing Buffered reader \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 /\* Constructor for accepting port \*/  
 public Addition\_looper(int port,Thread t){  
  
 this.port = port;  
 this.t = t;  
 }  
 public void run(){  
 try {  
 if(socket == null) {  
 socket = new DatagramSocket();  
 }  
  
 while(AsyncJokeClient.*sumFlag*) {  
 System.*out*.println("Enter numbers to sum:");  
 try {  
 AsyncJokeClient.*dataRec* = false;  
 /\* Accepting inputs from user console \*/  
 String input = in.readLine();  
 /\* fetching InetAddress \*/  
 InetAddress ip = InetAddress.*getLocalHost*();  
 /\* Converting the received inout to bytes \*/  
 buf = input.getBytes();  
 /\* Creating a Datagram packet to send the digits \*/  
 DatagramPacket dp = new DatagramPacket(buf, buf.length, ip, port);  
 /\* Sending the input to server \*/  
 socket.send(dp);  
 /\* Initializing a byte variable \*/  
 receive = new byte[1024];  
 /\* Datagram packet to receive data from server \*/  
 DatagramPacket dp1 = new DatagramPacket(receive, receive.length);  
 /\* Receiving the data \*/  
 socket.receive(dp1);  
 /\* Converting the data to string \*/  
 result = AsyncJokeClient.*data*(receive).toString().trim();  
 /\* Printing the result \*/  
 System.*out*.println("Result is " + result);  
 AsyncJokeClient.*dataRec* = true;  
  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 } catch (SocketException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

/\*\*\*\*\*\*\*\*\*\*\* Client Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea\_rt.jar=59784:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/satyayoganandaddala/Documents/Depaul\_Classes/CSC435/AsyncJoke\_Assignment/out/production/AsyncJoke\_Assignment AsyncJokeClient  
Satya Yoganand's Joke Client, 1.8.  
  
Using Server : localhost, Port: 9999  
Enter the User Name, (quit) to end:  
satya  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
2 4  
Result is 6  
Enter the numbers to sum :  
5 8  
Result is 13  
Joke Received  
JD satya: Why was six afraid of seven? Because seven eight nine.  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
3 5  
Result is 8  
Enter the numbers to sum :  
5 8  
Result is 13  
Enter the numbers to sum :  
4 7  
Result is 11  
Joke Received  
JA satya: What did one DNA say to the other DNA? ???Do these genes make me look fat????  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
1 1  
Result is 2  
Enter the numbers to sum :  
7 0  
Result is 7  
Joke Received  
JC satya: What do you get when you cross a polar bear with a seal? A polar bear.  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
6 6  
Result is 12  
Enter the numbers to sum :  
7 0  
Result is 7  
Joke Received  
JB satya: My IQ test results came back. They were negative.  
Joke Cycle Completed  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
2 5  
Result is 7  
Enter the numbers to sum :  
4 8  
Result is 12  
Proverb Received  
PA satya: Honesty is the best policy.  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
5 0  
Result is 5  
Proverb Received  
PB satya: Strike while the iron is hot.  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
6 6  
Result is 12  
Enter the numbers to sum :  
6 0  
Result is 6  
Proverb Received  
PD satya: An apple a day keeps the doctor away.  
Enter the User Name or Enter, (quit) to end:  
  
Enter the numbers to sum :  
4 9  
Result is 13  
Enter the numbers to sum :  
3 7  
Result is 10  
Proverb Received  
PC satya: Don???t judge a book by its cover.  
Proverb Cycle Completed ...  
Enter the User Name or Enter, (quit) to end:  
  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Server Log \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea\_rt.jar=59787:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/satyayoganandaddala/Documents/Depaul\_Classes/CSC435/AsyncJoke\_Assignment/out/production/AsyncJoke\_Assignment AsyncJokeServer  
  
Satya Yoganand's Joke server starting up...,listening at port 9999.  
  
Satya Yoganand's Admin Looper starting up...,listening at port : 5050.  
  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a joke ......  
Started Addition looper  
Printing received Digits : 2 4  
Creating new User...  
Printing received Digits : 5 8  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a joke ......  
Printing received Digits : 3 5  
Printing received Digits : 5 8  
UserId Already Exists !!!!  
Printing received Digits : 4 7  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a joke ......  
Printing received Digits : 1 1  
UserId Already Exists !!!!  
Printing received Digits : 7 0  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a joke ......  
Printing received Digits : 6 6  
UserId Already Exists !!!!  
Joke Cycle has been Completed ...  
Printing received Digits : 7 0  
Switching Mode to : proverb  
Mode set to Proverb  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a Proverb ......  
Printing received Digits : 2 5  
UserId Already Exists !!!!  
Printing received Digits : 4 8  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a Proverb ......  
UserId Already Exists !!!!  
Printing received Digits : 5 0  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a Proverb ......  
Printing received Digits : 6 6  
UserId Already Exists !!!!  
Printing received Digits : 6 0  
UserID,Name : b9a429cd-35b5-495e-a49b-89ccb8416106 satya  
Port Number : 59785, Address :Satyas-MacBook-Air.local/127.0.0.1  
UUID for the current Session : b9a429cd-35b5-495e-a49b-89ccb8416106  
Providing a Proverb ......  
Printing received Digits : 4 9  
UserId Already Exists !!!!  
Proverb Cycle has been Completed ...  
Printing received Digits : 3 7  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* JokeAdminClient Log\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea\_rt.jar=59790:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/satyayoganandaddala/Documents/Depaul\_Classes/CSC435/AsyncJoke\_Assignment/out/production/AsyncJoke\_Assignment AsyncJokeAdminClient  
Satya Yoganand's Client Admin, 1.8.  
  
Using Server : localhost, Port: 5050  
Enter Joke/Proverb to Switch Modes, (quit) to end:  
proverb  
Mode in Client Admin : proverb  
Enter Joke/Proverb to Switch Modes, (quit) to end: