/\*  
 1. Name / Date: satya Yoganand Addala / 22-09-2022  
  
 2. Java version used (java -version), if not the official version for the class: 17.0.2  
  
 3. Precise command-line compilation examples / instructions:  
 > javac JokeServer.java  
 > javac JokeClientAdmin.java  
 > javac JokeClient.java  
  
 4. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java JokeServer.java  
 > java JokeClient.java  
 > java JokeClientAdmin.java  
  
 All acceptable commands are displayed on the various consoles.  
  
 5. Notes:  
  
 \* This File Includes to use both primary and secondary servers on ports 4545,4546 respectively.  
 \* 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/probverbs respectively from the list.  
 \* Jokes have been taken from https://www.rd.com/list/funniest-one-liners-you-havent-heard-yet/  
 \* For each statement there is a check of which server the data has to go i.e., Primary or Secondary  
  
\*/  
  
import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintStream;  
import java.util.\*;  
  
public class JokeServer {  
 /\* 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 {  
 /\* Allowed number of connections from client to server at the same time \*/  
 int q\_len = 6;  
 /\* Initializing port to zero \*/  
 int port = 0;  
 /\* Initializing servsock to null \*/  
 ServerSocket servsock = null;  
 Socket sock;  
 if (args.length < 1) { /\* Checking for Arguments \*/  
 /\* Port used for primary Server \*/  
 port = 4545;  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 servsock = new ServerSocket(port, q\_len);  
 System.*out*.println("Satya Yoganand's Joke server starting up...,listening at port 4545.\n");  
 }  
 else if(args.length == 1 && args[0].equals("secondary")){  
 /\* Port used for making secondary server connection \*/  
 port = 4546;  
 /\* Setting the SeverType to Secondary Server \*/  
 *serverType* = true;  
 /\* The ServerSocket will take the portNumber and queue length on which it is going to communicate \*/  
 servsock = new ServerSocket(port, q\_len);  
 System.*out*.println("<S2> Satya Yoganand's Secondary Joke server starting up...,listening at port 4546.\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();  
  
  
 while (true) { /\* Waiting to accept all the connections \*/  
 /\* This is an accept method for listening from the client and is also a blocking call \*/  
 sock = servsock.accept();  
 /\* This Start method will invoke the run method in Worker class \*/  
 new Worker(sock).start();  
 }  
 }  
}  
  
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;  
 if(!JokeServer.*serverType*) { /\* Checking ServerType \*/  
 /\* Using port 5050 for Admin Client \*/  
 port = 5050;  
 System.*out*.println("Satya Yoganand's Admin Looper starting up...,listening at port : "+port +".\n");  
 }  
 else{  
 /\* Using port 5051 for Admin Client in secondary server \*/  
 port = 5051;  
 System.*out*.println("<S2> Satya Yoganand's Admin Looper starting up...,listening at port : "+port +".\n");  
 }  
 /\* Creating Socket Variable \*/  
 Socket sock;  
 try {  
 ServerSocket servsock = new ServerSocket(port, q\_len);  
 while (*adminControlSwitch*) {  
 /\* Accepting the connections \*/  
 sock = servsock.accept();  
 /\* Starting the ModeWorker \*/  
 new ModeWorker(sock).start();  
 }  
 } catch (IOException ioe) {  
 System.*out*.println(ioe);  
 }  
 }  
}  
  
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(!JokeServer.*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 \*/  
 JokeServer.*mode* = "Joke";  
 /\* Checking the serverType \*/  
 if(!JokeServer.*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 \*/  
 JokeServer.*mode* = "Proverb";  
 /\* Checking the server Type \*/  
 if(!JokeServer.*serverType*) {System.*out*.println("Mode set to Proverb");}  
 else {System.*out*.println("<S2> Mode set to Proverb");}  
 }  
 else {  
 /\* Checking the serverType \*/  
 if(!JokeServer.*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(!JokeServer.*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;  
  
 /\* Constructor for fetching the Socket \*/  
 Worker(Socket s) {  
 *sock* = s;  
 }  
  
 public void run() {  
  
 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 name;  
 String userId;  
  
 /\* Blocking Call - the server code will pauses at this point and starts listening to the input from client \*/  
 /\* Reading name from Client \*/  
 name = *in*.readLine();  
 /\* Reading Unique ID from Client \*/  
 userId = *in*.readLine();  
 if(!JokeServer.*serverType*) { /\* Checking Server Type \*/  
 System.*out*.println("UUID for the current Session : " + userId);  
 System.*out*.println("Providing a "+ JokeServer.*mode* +" ......");  
 }  
 else{  
 System.*out*.println("<S2> UUID for the current Session : " + userId);  
 System.*out*.println("<S2> Providing a "+ JokeServer.*mode* +" ......");  
 }  
 /\* Invoking RandomSelection method for generating random Jokes/Proverbs \*/  
 *randomSelection*(userId,name,*out*);  
  
 } catch (IOException x) { /\* Handling the IOException if the name variable is not valid \*/  
 if(!JokeServer.*serverType*){System.*out*.println("Server read error");}  
 else{System.*out*.println("<S2> Server read error");}  
 x.printStackTrace();  
 }  
 /\* Closing the socket connection after the process is done \*/  
 *sock*.close();  
 } catch (IOException ioe) {  
 System.*out*.println(ioe);  
 }  
 }  
  
 static void randomSelection(String userId,String name,PrintStream output) throws IOException {  
 *user\_Name* = name;  
 String Joke\_Status = null;  
 /\* Checking for the userId in UUID List \*/  
 if(*UUIDList*.contains(userId)){  
 if(!JokeServer.*serverType*) { /\* Checking Server Type \*/  
 System.*out*.println("UserId Already Exists !!!!");  
 }  
 else{  
 System.*out*.println("<S2> UserId Already Exists !!!!");  
 }  
  
 if(JokeServer.*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(!JokeServer.*serverType*) { /\* Checking Server Type \*/  
 /\* Writing random Joke to Client \*/  
 output.println(random\_Joke);  
 if (jokesCopyListSize == 0) {  
 output.println("Joke Cycle Completed ... ");  
 }  
 }  
 else {  
 output.println("<S2> "+ random\_Joke);  
 if(jokesCopyListSize == 0){output.println("<S2> Joke Cycle Completed ... ");}  
 }  
  
 }  
 /\* Server mode is set to Proverb \*/  
 else if(JokeServer.*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(!JokeServer.*serverType*) { /\* Checking Server Type \*/  
 /\* Writing random proverb to Client \*/  
 output.println(random\_Proverb);  
 if(proverbCopyListSize == 0 ){output.println("Proverb Cycle Completed ... ");}  
 }  
 else {  
 output.println("<S2> "+ random\_Proverb);  
 if(proverbCopyListSize == 0 ){output.println("<S2> Proverb Cycle Completed ... ");}  
 }  
 }  
 }  
 else {  
 if(!JokeServer.*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(JokeServer.*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(!JokeServer.*serverType*) /\* Checking Server Type \*/  
 output.println(random\_Joke);  
 else output.println("<S2> "+ random\_Joke);  
 }  
 else if(JokeServer.*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(!JokeServer.*serverType*) /\* Checking Server Type \*/  
 /\* printing random proverb in primary server \*/  
 output.println(random\_Proverb);  
 else output.println("<S2> "+ random\_Proverb); /\* printing random proverb in secondary server \*/  
 }  
 }  
  
 }  
  
 /\* 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(!JokeServer.*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(!JokeServer.*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);  
 }  
  
 }  
 }  
}

/\*  
\* 1. Name / Date: satya Yoganand Addala / 22-09-2-22  
  
 2. Java version used (java -version), if not the official version for the class: 17.0.2  
  
 3. Precise command-line compilation examples / instructions:  
 > javac JokeServer.java  
 > javac JokeClientAdmin.java  
 > javac JokeClient.java  
  
 4. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java JokeServer.java  
 > java JokeClient.java  
 > java JokeClientAdmin.java  
  
 All acceptable commands are displayed on the various consoles.  
  
 5. Notes:  
  
 \* Running this file will ask the user to enter a username and send it to Server and after Clicking enter the Server will give Joke or Proverb Accordingly.  
 \* Generating Unique Id through UUID.randomUUID() and storing it in Server.  
 \* For each statement there is a check of which server the data has to go i.e., Primary or Secondary  
  
\*/  
  
import java.io.\*;  
import java.net.\*;  
import java.util.UUID;  
  
public class JokeClient {  
 public static void main(String args[]) throws IOException {  
 //The Server used for connection is a localhost  
 String serverName;  
 boolean serverType = false;  
 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: 4545");  
 }  
 else if(args.length == 1 && args[0].equals("secondary")){  
 /\* Setting the Severname to localhost \*/  
 serverName = "localhost";  
 /\* Setting the SeverType to Secondary Server \*/  
 serverType = true;  
 System.*out*.println("<S2> Satya Yoganand's Secondary Joke Client, 1.8.\n");  
 System.*out*.println("<S2> Using Server : " + serverName + ", Port: 4546");  
 }  
 else {  
 serverName = args[0];  
 }  
  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 if(!serverType) { /\* Checking ServerType \*/  
 System.*out*.println("Enter the User Name, (quit) to end: ");  
 }  
 else{  
 System.*out*.println("<S2> Enter the User Name, (quit) to end: ");  
 }  
 System.*out*.flush();  
 String name;  
 name = in.readLine();  
 while(name.equals("")){  
 if(!serverType){  
 System.*out*.println("Please enter a valid name : ");  
 }  
 else{System.*out*.println("<S2> Please enter a valid name : ");}  
  
 name = in.readLine();  
 }  
 try {  
 String nextLine;  
 /\* Generating a random UUID for each client \*/  
 String userId = UUID.*randomUUID*().toString();  
  
 do {  
 /\* 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*(serverType,userId, name, serverName);  
 } 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(boolean sType,String userId,String name,String serverName){  
 /\* Creating a Socket variable \*/  
 Socket sock;  
 /\* Creating a BufferedReader variable to send output from client to server \*/  
 BufferedReader from\_Server;  
 /\* Creating a PrintStream variable to receive input from server \*/  
 PrintStream to\_Server;  
 /\* Creating a variable for receiving text from server \*/  
 String textFromServer;  
 /\* Variable for receiving joke status from server \*/  
 String Joke\_Status;  
  
 try{  
// sock = new Socket(serverName,4545);  
 if(!sType){  
 //connecting to server @port 4545 and servername  
 sock =new Socket(serverName,4545);  
 }  
 else{  
 //connecting to server @port 4546 and servername  
 sock =new Socket(serverName,4546);  
 }  
 /\* Fetching the inputs for the server using BufferedReader through inputStream \*/  
 from\_Server = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* used to send output from the client using PrintStream \*/  
 to\_Server = new PrintStream(sock.getOutputStream());  
 /\* Sending Username to JokeServer \*/  
 to\_Server.println(name);  
 /\* Sending Unique Userid to JokeServer \*/  
 to\_Server.println(userId);  
 /\* Used for sending data immediately to server \*/  
 to\_Server.flush();  
 /\* Reading data from server \*/  
 textFromServer = from\_Server.readLine();  
 if(textFromServer != null) System.*out*.println(textFromServer);  
 /\* Reading Joke Status from Server \*/  
 Joke\_Status = from\_Server.readLine();  
 if(Joke\_Status != null) System.*out*.println(Joke\_Status);  
 /\* Close the Socket after the Connection is done \*/  
 sock.close();  
  
 } catch (UnknownHostException e) { /\* Catch block for handling UnknownHostException \*/  
 throw new RuntimeException(e);  
 } catch (IOException e) { /\* Catch block for handling IOException \*/  
 throw new RuntimeException(e);  
 }  
 }  
}

/\*  
 1. Name / Date: satya Yoganand Addala / 22-09-2022  
  
 2. Java version used (java -version), if not the official version for the class: 17.0.2  
  
 3. Precise command-line compilation examples / instructions:  
 > javac JokeServer.java  
 > javac JokeClientAdmin.java  
 > javac JokeClient.java  
  
 4. Precise examples / instructions to run this program:  
  
 In separate shell windows run all the below commands :  
  
 > java JokeServer.java  
 > java JokeClient.java  
 > java JokeClientAdmin.java  
  
 All acceptable commands are displayed on the various consoles.  
  
 5. Notes:  
  
 \* Running this file will toggle between the Jokes/Proverbs.  
 \* For this the user has to enter the word Joke/Proverb for each toggle.  
 \* For each statement there is a check of which server the data has to go i.e., Primary or Secondary  
  
\*/  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintStream;  
import java.net.Socket;  
  
public class JokeClientAdmin {  
 private static String *mode*;  
 public static void main(String args[]) throws IOException {  
  
 /\* servername used for connection \*/  
 String serverName;  
 /\* Setting the ServerType to primary by default\*/  
 boolean serverType = false;  
 if (args.length < 1) {  
 /\* Setting the Severname to localhost \*/  
 serverName = "localhost";  
 System.*out*.println("Satya Yoganand's Client Admin, 1.8.\n");  
 System.*out*.println("Using Server : " + serverName + ", Port: 5050");  
 }  
 else if(args.length == 1 && args[0].equals("secondary")){  
 /\* Setting the Severname to localhost \*/  
 serverName = "localhost";  
 /\* Setting the SeverType to Secondary Server \*/  
 serverType = true;  
 System.*out*.println("<S2> Satya Yoganand's Secondary Client Admin, 1.8.\n");  
 System.*out*.println("<S2> Using Server : " + serverName + ", Port: 5051");  
 }  
 else {  
 serverName = args[0];  
 }  
 /\* Buffered Reader for reading Inputs \*/  
 BufferedReader in = new BufferedReader(new InputStreamReader(System.*in*));  
 /\* Variable for Server Mode whether it is a Joke / Proverb \*/  
  
  
 do {  
 if(!serverType) { /\* Checking ServerType \*/  
 System.*out*.println("Enter Joke/Proverb to Switch Modes, (quit) to end: ");  
 }  
 else{System.*out*.println("<S2> Enter Joke/Proverb to Switch Modes, (quit) to end: ");}  
 /\* This is for sending data Immediately \*/  
 System.*out*.flush();  
 /\* Enter the word Joke/Proverb to set the mode for primary/secondary server \*/  
 *mode* = in.readLine();  
 if(!serverType) { /\* Checking ServerType \*/  
 System.*out*.println("Mode in Client Admin : " + *mode*);  
 }  
 else{System.*out*.println("<S2> Mode in Client Admin : " + *mode*);}  
 /\* If Entered quit then the process gets stopped or else it will fetch the HostName and HostIp \*/  
 if (*mode*.indexOf("quit") < 0)  
 *modeControl*(serverType,serverName,*mode*);  
 } while (*mode*.indexOf("quit") < 0);  
  
 }  
  
 public static void modeControl(boolean sType,String sName, String sMode) {  
 /\* Creating the Socket Variable \*/  
 Socket sock;  
 /\* Creating the variable for BufferedReader \*/  
 BufferedReader fromServer;  
 /\* Creating the variable for PrintStream \*/  
 PrintStream toServer;  
 /\* Variable to read text from Server \*/  
 String textFromServer;  
 try{  
 if(!sType){ /\* Checking ServerType \*/  
 /\* connecting to server @port 5050 and servername \*/  
 sock =new Socket(sName,5050);  
 }  
 else{  
 /\* connecting to server @port 5051 and servername \*/  
 sock =new Socket(sName,5051);  
 }  
  
 /\* Creating Buffered reader to store the inputs from Server \*/  
 fromServer=new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 /\* Creating PrintStream to send data to the server \*/  
 toServer=new PrintStream(sock.getOutputStream());  
  
 /\* Sending Mode to the Server \*/  
 toServer.println(sMode);  
 /\* sending the server mode to the server immediately \*/  
 toServer.flush();  
 /\* Reading Input from Server \*/  
 textFromServer=fromServer.readLine();  
 /\* Condition to check if the text for server is null \*/  
 if(textFromServer !=null) System.*out*.println(textFromServer);  
 /\* Closing the socket once the task is done \*/  
 sock.close();  
 }catch (IOException x) { /\* Catch for capturing the IOException \*/  
 System.*out*.println("socket error");  
 x.printStackTrace();  
 }  
 }  
  
 public static void toggleMode(int mode){  
 if(mode == 0){  
 mode =1;  
 }  
 else{mode = 0;}  
 }  
}