/\*--------------------------------------------------------

Urvi Patel

Due date: 1/22/2017

Java version: 1.8

Command-line compilation examples / instructions:

To compile, place .java files in same directory and run:

> javac JokeServer.java JokeCient.java JokeClientAdmin.java

or if these are the only files in the directory:

> javac \*.java

4. Precise examples / instructions to run this program:

The JokeServer, JokeClient, and JokeClientAdmin may run on the same machine

or on different machines. For illustration purposes, assume that the JokeServer

runs on 140.192.1.23

If the JokeServer runs primary server (only) and JokeClient is and JokeClientAdmin

are on same machine as JokeServer. That is, on command lines args for JokeClient

and JokeClientAdmin assume that the server is running on localhost

In separate shell windows, type the following:

> java JokeServer

> java JokeClient

> java JokeClientAdmin

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following:

> java JokeServer

> java JokeClient 140.192.1.23

> java JokeAdminClient

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following::

> java JokeServer secondary

> java JokeClient 140.192.1.23

> java JokeAdminClient

List of files needed for running the program.

b. JokeServer.java

c. JokeClient.java

d. JokeClientAdmin.java

Notes:

Used Java class Random to generate random numbers from 0 - 3 to get random

indexes into joke/proverb array.

----------------------------------------------------------\*/

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.Socket;

import java.util.UUID;

public class JokeClient

{

private static String primaryServerMachine = "";

private static String secondaryServerMachine = "";

private static final int PRIMARY\_PORT = 4545;

private static final int SECONDARY\_PORT = 4546;

private static final UUID PRIMARY\_UUID = UUID.randomUUID();

private static final UUID SECONDARY\_UUID = UUID.randomUUID();

public static void main(String args[])

{

String userInput;

if(args.length == 0)

{

primaryServerMachine = "localhost";

}

if(args.length == 1)

{

primaryServerMachine = args[0];

}

if(args.length == 2)

{

primaryServerMachine = args[0];

secondaryServerMachine = args[1];

}

/\*Print informative messages to the user, reiterating his/her choices for

\* a primary server and (if applicable) a secondary server\*/

System.out.println("Starting Joke Client, 1.8\n");

if(args.length < 2)

{

System.out.println("Using server: " + primaryServerMachine + ", Port: " + PRIMARY\_PORT);

}

else

{

System.out.println("Using server: " + primaryServerMachine + ", Port: " + PRIMARY\_PORT);

System.out.println("Using server: " + secondaryServerMachine + ", Port: " + SECONDARY\_PORT);

}

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

boolean useSecondary = false;

/\* Generate an id for the user to send to the server so server can

\* can check which messages have already been sent to this client \*/

try

{

/\* Get the user name from the user, store it in a variable and send

\* it to the server. The server will then insert it into the joke

\* or proverb that it sends to the client. \*/

String userName;

System.out.print("Enter your name: ");

userName = in.readLine();

System.out.flush();

/\* Have user hit <Enter> key to get more jokes or proverbs,

\* or type 'quit' to exit \*/

do

{

userInput = in.readLine();

if(userInput.indexOf("quit") < 0)

{

if(userInput.equals("s"))

{

if(secondaryServerMachine.isEmpty())

{

System.out.println("No secondary server, using " + primaryServerMachine + " on " + PRIMARY\_PORT);

}

else

{

//System.out.println("useSecondary before toggle " + useSecondary);

useSecondary = !useSecondary;

//System.out.println("useSecondary after toggle " + useSecondary);

String serverUsed = useSecondary ? secondaryServerMachine : primaryServerMachine;

int portUsed = useSecondary ? SECONDARY\_PORT : PRIMARY\_PORT;

System.out.println("Using " + serverUsed + " on port " + portUsed);

}

}

//System.out.println("Calling getJoke() with useSecondary as " + useSecondary);

getJoke(userName, useSecondary);

}

}while(userInput.indexOf("quit") < 0);

System.out.println("Goodbye, " + userName + "!");

}

catch(IOException ioe)

{

ioe.printStackTrace();

}

}//End main

static void getJoke(String name, boolean secondary)

{

Socket socket;

BufferedReader fromServer;

PrintStream toServer;

String textFromServer;

String serverName = "";

int port = 0;

UUID id = null;

if(secondary)

{

serverName = secondaryServerMachine;

port = SECONDARY\_PORT;

id = SECONDARY\_UUID;

}

else

{

serverName = primaryServerMachine;

port = PRIMARY\_PORT;

id = PRIMARY\_UUID;

}

try

{

socket = new Socket(serverName, port); // Create a socket to the server, listening port 4545

fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Input stream to read data from server

toServer = new PrintStream(socket.getOutputStream()); // Output stream on which to place data to send to server

toServer.println(name); // Send the user name to the server

toServer.flush();

toServer.println(id); // Send the id to the server

while( (textFromServer = fromServer.readLine()) != null)

{

System.out.println(textFromServer); // Print out joke/proverb received from server

}

socket.close(); // Close the socket connection for this client to prevent resource leak

}

catch(IOException e)

{

System.out.println("Socket error"); // Catch exceptions related to writing to or reading from socket

e.printStackTrace();

}

}//End getJoke()

/\* static void getJoke(String name, String serverName, UUID id)

{

Socket socket;

BufferedReader fromServer;

PrintStream toServer;

String textFromServer;

try

{

socket = new Socket(serverName, PORT); // Create a socket to the server, listening port 4545

fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Input stream to read data from server

toServer = new PrintStream(socket.getOutputStream()); // Output stream on which to place data to send to server

toServer.println(name); // Send the user name to the server

toServer.flush();

toServer.println(id); // Send the id to the server

while( (textFromServer = fromServer.readLine()) != null)

{

System.out.println(textFromServer); // Print out joke/proverb received from server

}

socket.close(); // Close the socket connection for this client to prevent resource leak

}

catch(IOException e)

{

System.out.println("Socket error"); // Catch exceptions related to writing to or reading from socket

e.printStackTrace();

}

}//End getJoke() \*/

}// End class JokeClient

/\*--------------------------------------------------------

Urvi Patel

Due date: 1/22/2017

Java version: 1.8

Command-line compilation examples / instructions:

To compile, place .java files in same directory and run:

> javac JokeServer.java JokeCient.java JokeClientAdmin.java

or if these are the only files in the directory:

> javac \*.java

4. Precise examples / instructions to run this program:

The JokeServer, JokeClient, and JokeClientAdmin may run on the same machine

or on different machines. For illustration purposes, assume that the JokeServer

runs on 140.192.1.23

If the JokeServer runs primary server (only) and JokeClient is and JokeClientAdmin

are on same machine as JokeServer. That is, on command lines args for JokeClient

and JokeClientAdmin assume that the server is running on localhost

In separate shell windows, type the following:

> java JokeServer

> java JokeClient

> java JokeClientAdmin

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following:

> java JokeServer

> java JokeClient 140.192.1.23

> java JokeAdminClient

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following::

> java JokeServer secondary

> java JokeClient 140.192.1.23

> java JokeAdminClient

List of files needed for running the program.

b. JokeServer.java

c. JokeClient.java

d. JokeClientAdmin.java

Notes:

Used Java class Random to generate random numbers from 0 - 3 to get random

indexes into joke/proverb array.

----------------------------------------------------------\*/

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 primaryServerMachine = "";

private static String secondaryServerMachine = "";

private static final int PRIMARY\_PORT = 5050;

private static final int SECONDARY\_PORT = 5051;

public static void main(String args[])

{

String inputFromUser;

/\*When the client is started on the command line, an argument may be given to denote

the machine to which the client will connect. If no argument is given, the client

will connect to localhost. If one argument is given, the client will connect to

that IP. If two arguments are given, the first is considered to be the IP address

of primary server and the second argument is the IP of the secondary server\*/

if(args.length == 0)

{

primaryServerMachine = "localhost";

}

if(args.length == 1)

{

primaryServerMachine = args[0];

}

if(args.length == 2)

{

primaryServerMachine = args[0];

secondaryServerMachine = args[1];

}

/\* Print informative messages to the user, reiterating his/her choices for

\* a primary server and (if applicable) a secondary server\*/

System.out.println("Starting Joke Admin Client, 1.8\n");

if(args.length < 2)

{

System.out.println("Primary server: " + primaryServerMachine + ", Port: " + PRIMARY\_PORT);

}

else

{

System.out.println("Primary server: " + primaryServerMachine + ", Port: " + PRIMARY\_PORT);

System.out.println("Secondary server: " + secondaryServerMachine + ", Port: " + SECONDARY\_PORT);

}

/\*Create a BufferedReader object to read the <enter> entered

by the user from the keyboard (i.e., System.in) \*/

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

boolean useSecondary = false; // Start out using the primary server

String primaryOrSecondary = "";

String serverUsed = "";

int portUsed;

try

{

do

{

inputFromUser = in.readLine(); // Read in input from a user, only 'quit' will exit loop; any other key will toggle mode

if(inputFromUser.indexOf("quit") < 0)

{

if(inputFromUser.equals("s")) // Input of 's' toggles user between primary and secondary server

{

if(secondaryServerMachine.isEmpty()) // If no secondary server is given on command line, inform the user and user primary

{

System.out.println("No secondary server, using " + primaryServerMachine + " on " + PRIMARY\_PORT);

}

else

{

useSecondary = !useSecondary; // Toggle between primary and secondary

/\* Display message to user which server/port they are on

based on the value of useSecondary variable, which is

toggled upon user input \*/

primaryOrSecondary = useSecondary ? "SECONDARY" : "PRIMARY";

serverUsed = useSecondary ? secondaryServerMachine : primaryServerMachine;

portUsed = useSecondary ? SECONDARY\_PORT : PRIMARY\_PORT;

System.out.println("Using " + primaryOrSecondary + " server " + serverUsed + " on port " + portUsed);

continue; // Print message to user and continue, user can then hit <enter> to change mode or 's' toggle servers

}

}

sendMode(useSecondary);

}

}while(inputFromUser.indexOf("quit") < 0); // Keep looping until user types 'quit'

System.out.println("Goodbye!");

}

catch(IOException ioe)

{

ioe.printStackTrace();

}

}//End main

/\* sendMode() takes one parameter, a boolean, which denotes

\* whether the mode is to be sent to the primary or secondary

\* server.

\*

\* Based on the value of the 'secondary' variable passed in,

\* the appropriate server and port are set

\*

\* A socket, input stream and output stream are created

\*

\* The socket connects to the server and port based on the

\* value of the 'secondary' variable passed in\*/

static void sendMode(boolean secondary)

{

Socket socket;

PrintStream toServer;

BufferedReader fromServer;

String serverName = "";

int port = 0;

String mode = "";

// Set machine and port to point to secondary server

if(secondary)

{

serverName = secondaryServerMachine;

port = SECONDARY\_PORT;

}

// Set machine and port to point to primary server

else

{

serverName = primaryServerMachine;

port = PRIMARY\_PORT;

}

try

{

socket = new Socket(serverName, port); // Connect to server via socket on server and port, determined above

toServer = new PrintStream(socket.getOutputStream()); // Stream used to send data to the server

fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream())); // Stream used to read data from server

String currMode = fromServer.readLine(); //Get current mode from server

mode = currMode.equals("joke-mode") ? "proverb-mode" : "joke-mode"; // Toggle the current mode

toServer.println(mode); // Send the mode to the server on the output stream

toServer.flush();

String responseFromServer = fromServer.readLine(); // Get a response from the server from the input stream

System.out.println(responseFromServer); // Print it out to the AdminClient user

socket.close(); // Close the connection to prevent resource leak

}

catch(IOException e)

{

System.out.println("Socket error");

e.printStackTrace();

}

}//End sendMode()

}

/\*--------------------------------------------------------

Urvi Patel

Due date: 1/22/2017

Java version: 1.8

Command-line compilation examples / instructions:

To compile, place .java files in same directory and run:

> javac JokeServer.java JokeCient.java JokeClientAdmin.java

or if these are the only files in the directory:

> javac \*.java

4. Precise examples / instructions to run this program:

The JokeServer, JokeClient, and JokeClientAdmin may run on the same machine

or on different machines. For illustration purposes, assume that the JokeServer

runs on 140.192.1.23

If the JokeServer runs primary server (only) and JokeClient is and JokeClientAdmin

are on same machine as JokeServer. That is, on command lines args for JokeClient

and JokeClientAdmin assume that the server is running on localhost

In separate shell windows, type the following:

> java JokeServer

> java JokeClient

> java JokeClientAdmin

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following:

> java JokeServer

> java JokeClient 140.192.1.23

> java JokeAdminClient

If the JokeServer runs on primary and JokeClient or JokeClientAdmin are running on

another machine then you must put the IP address of the server to the command line.

For example, if the JokeClient is on a different machine from the JokeServer and the

JokeAdminClient is on the same machine as the JokeServer, then in separate shell

windows, type the following::

> java JokeServer secondary

> java JokeClient 140.192.1.23

> java JokeAdminClient

List of files needed for running the program.

b. JokeServer.java

c. JokeClient.java

d. JokeClientAdmin.java

Notes:

Used Java class Random to generate random numbers from 0 - 3 to get random

indexes into joke/proverb array.

----------------------------------------------------------\*/

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.util.Random;

class Worker extends Thread

{

Socket socket;

String joke;

String[] jokes = new String[4];

String[] proverbs = new String[4];

static int jIndex;

static int pIndex;

ArrayList<String> jokePrefixList = new ArrayList<>();

ArrayList<String> proverbPrefixList = new ArrayList<>();

Worker (Socket sock)

{

socket = sock;

initializeJokesAndProverbsLists();

initializeJokesAndProverbsPrefixLists();

}

/\* These ArrayLists are used to help keep track of the jokes/proverbs that have

\* already been sent to the client. The jokes/proverbs are saved in separate

\* hash tables where a UUID (provided by the user) is the key and the value is

\* a list of jokes/proverbs for that user. The approach taken was to loop

\* through the list of jokes/proverbs for a given user and pull off the prefixes

\* and store them in a list. Then, for every prefix in this list, use the

\* indexOf() method on the jokePrefixArray and proverbPrefixArray to

\* find the indexes for the jokes/proverbs that have already been used.

\*

\* For example, assume client was sent jokes B and C:

\*

\* Access the user's entry in the joke table using the

\* id sent by the client.

\*

\* Loop through the list of jokes for this client, pull off the

\* prefixes for jokes B and C. The list would look like => [JB, JC]

\*

\* Use the fact that the first joke in the joke array is prefixed

\* with JA, the second with JB, etc., you can use the indexOf

\* method on the jokePrefixArray to find the corresponding index

\* into the joke array. In this case, call to indexOf for JB would

\* give 1, and that means that jokes[1] has already been sent to

\* the client.

\*

\* A similar example would apply for proverbs.

\* \*/

private void initializeJokesAndProverbsPrefixLists()

{

jokePrefixList.add("JA");

jokePrefixList.add("JB");

jokePrefixList.add("JC");

jokePrefixList.add("JD");

proverbPrefixList.add("PA");

proverbPrefixList.add("PB");

proverbPrefixList.add("PC");

proverbPrefixList.add("PD");

}

/\* This method populates the lists that will hold jokes and proverbs,

\* in order. That is, the first entry holds joke A, the

\* second hold joke B, etc. Since you don't want to send

\* the same joke to the client until all 4 jokes are sent,

\* you need to keep track of which jokes are "valid" to send

\* to the user. This arrangement makes it simple to know

\* the valid indexes in this array once you learn which

\* joke(s) have already been sent to the client. For

\* example, if you know that joke A has been sent, then

\* you know that the first index is not valid.

\*

\* The "<#name>" is used as a placeholder so that the uer's

\* name can be inserted into the jokes and proverbs sent

\* from the server.\*/

private void initializeJokesAndProverbsLists()

{

jokes[0] = "JA <#name>: A recent scientific study showed that out of 2,293,618,367 people, 94% are too lazy to actually read that number.";

jokes[1] = "JB <#name>: What time did the man go to the dentist? Tooth hurt-y.”";

jokes[2] = "JC <#name>: Why did the Clydesdale give the pony a glass of water? Because he was a little horse!";

jokes[3] = "JD <#name>: I had a dream that I was a muffler last night. I woke up exhausted!";

proverbs[0] = "PA <#name>: There is no such thing as a free lunch";

proverbs[1] = "PB <#name>: Early to bed, early to rise";

proverbs[2] = "PC <#name>: A picture is worth a thousand words";

proverbs[3] = "PD <#name>: Absence makes the heart grow fonder";

}

public void run()

{

BufferedReader inputStream = null;

PrintStream outputStream = null;

try

{

/\*Create input and output streams to read data from and write data to the socket,

\* thus allowing communication between server and client.

\* The underlying input stream that is read from is the socket's input stream.

\* The underlying output stream that is written to is the socket's output stream \*/

inputStream = new BufferedReader(new InputStreamReader(socket.getInputStream()));

outputStream = new PrintStream(socket.getOutputStream());

try

{

// Get the user's name sent from the client and put it in a variable

String user = inputStream.readLine();

//Get the id sent from the user and put it in a variable

String id = inputStream.readLine();

/\* Default mode is joke-mode, and any string other than "proverb-mode" or

\* "quit" is taken to be "joke-mode\*/

if(JokeServer.serverMode.equals("proverb-mode"))

{

sendMessage(id, user, outputStream, proverbs, JokeServer.proverbMap, proverbPrefixList);

}

else //Server is in joke-mode

{

sendMessage(id, user, outputStream, jokes, JokeServer.jokeMap, jokePrefixList);

}

}//End inner try

catch(IOException ex)

{

System.out.println("Server read error"); //Exception trying to read from input stream or write to output stream

ex.printStackTrace();

}

socket.close();

}//End outer try

catch(IOException e)

{

System.out.println(e); //Exception creating the input/output streams

}

}

/\* This method takes the user's name, user id, the output stream for the server, an array

\* that contains jokes or proverbs, and a hash map that consists of the jokes or proverbs

\* that were sent to the user with the given id

\*

\* The method finds a joke or proverb (depending on the hash map) that has not yet been

\* sent to the user and sends it.

\*/

public void sendMessage(String id, String user, PrintStream out, String[] messageArray, Map<String, ArrayList<String>> map, ArrayList<String> prefixList)

{

String messageFromServer = "";

/\* Holds the indexes for jokes[] or proverbs[] array for the jokes or proverbs that have

\* already been sent to the client \*/

ArrayList<Integer> existingIndexes = new ArrayList<>();

// Check if the user has connected before

if(map.get(id) == null)

{

/\* If not, create an entry for the user in the appropriate hash map

\* with the user's id as the key and an empty list \*/

map.put(id, new ArrayList<String>());

/\* Since the user has not connected and he/she has not received any

\* thing from the server as of yet. This means that any joke or

\* proverb is valid. The message array contains 4 items, so get a

\* number from 0 to 3 and get the joke or proverb in the joke or proverb array

\* at that index. Then, send it to the client on the output stream.\*/

Random pRandom = new Random();

int pIndex = pRandom.nextInt(4);

messageFromServer = messageArray[pIndex];

messageFromServer = messageFromServer.replaceAll("<#name>", user);

out.println(messageFromServer);

//Add the joke or proverb to the user's list of jokes or proverbs that have been sent

map.get(id).add(messageFromServer);

//printMap(map, user);

}

else //The else statement is hit when the user has connected previously

{

/\* Get the jokes or proverbs that have already been sent to this user by

looking them up in the appropriate hash map\*/

ArrayList<String> msgs = map.get(id);

/\* Get the indexes into to the joke or proverb array for the

\* jokes or proverbs that have already been sent to the user \*/

existingIndexes = getIndexesOfPriorMessages(id, map, prefixList);

/\* Get a random number between 0 and 3 and if that the

\* generated number is in the list of indexes that

\* represent jokes/proverbs already sent to the user, keep getting

\* a random number until you get one that is not in the list of

\* indexes for jokes or proverbs already sent to the client\*/

Random r1 = new Random();

int randomIndex1 = r1.nextInt(4);

while(existingIndexes.contains(randomIndex1))

{

randomIndex1 = r1.nextInt(4);

}

/\* Once an index if found, use it to index into the jokes or proverbs

\* array to find a joke or proverb that has not yet been sent to the

\* client, insert the user name into the joke or proverb and send the

\* joke or proverb to the client.

\*

\* Lastly, add that joke or proverb to the list of jokes or proverbs for

\* that user id \*/

messageFromServer = messageArray[randomIndex1];

messageFromServer = messageFromServer.replaceAll("<#name>", user);

out.println(messageFromServer);

msgs.add(messageFromServer);

//printMap(map, user);

if(msgs.size() == 4)

{

msgs.clear();

}

}

}// End sendMessage()

/\* This method takes the user id, a hash map of jokes or proverbs, and

\* a list prefixes for jokes or provers, i.e., [JA, JC] or [PA, PD]

\*

\* This method returns a list of integers that are indexes of the jokes

\* or proverb array for the jokes or proverbs that have already been sent

\* to the user

\*

\* It first gets the list of jokes or proverbs (depending on the hashmap that

\* has been passed in) that have already been sent to this user. This is

\* accomplished by doing a lookup of the hash table, using the passed-in id as

\* the key.

\*

\* Then, for each joke in the list, it pulls off the the prefix,

\* for example, JA, JC, and JD and uses indexOf() on the jokePrefixArray

\* to find the corresponding index for each prefix.

\*

\* For the example above, for JA, JC, and JD, the method would return

\* the list [0, 2, 3]

\*/

ArrayList<Integer> getIndexesOfPriorMessages(String id, Map<String, ArrayList<String>> map, ArrayList<String> prefixes)

{

ArrayList<String> msgs = map.get(id);

ArrayList<Integer> mIndexes = new ArrayList<>();

for(String s : msgs)

{

String prefix = s.substring(0, 2);

mIndexes.add(prefixes.indexOf(prefix));

}

return mIndexes;

}

//Print out the joke or proverb hash map to verify things are working as expected

void printMap(Map<String, ArrayList<String>> m, String uName)

{

System.out.println("\nMap for: " + uName);

for(String k:m.keySet())

{

ArrayList<String> value = m.get(k);

System.out.print(k + ":");

for(String s:value)

{

s = s.substring(0, 2);

System.out.print(s + "\t");

}

}

}

} //End class Worker

public class JokeServer

{

public static boolean controlSwitch = true; // Controls the loop where the server is waiting. Set to true for infinite loop

public static String serverMode = "joke-mode"; // Default mode is joke-mode

/\* Hash maps to store the jokes and proverbs for each client (user). The hash maps

\* use the UUID sent from the client as a key and the value is the list of jokes

\* or proverbs that have been sent to that client \*/

public static Map<String, ArrayList<String>> jokeMap = new HashMap<>();

public static Map<String, ArrayList<String>> proverbMap = new HashMap<>();

public static void main(String a[]) throws IOException

{

int q\_len = 6;

int port = 4545;

Socket socket;

ServerSocket serverSocket = new ServerSocket(port, q\_len);

/\* Since you cannot block twice in the same thread at the same time, one of the calls needs

\* to be asynchronous. To this end, an asynchronous call is made to wait for AdminClient

\* input before blocking for client (regular user) input \*/

JokeAdminServer jokeAdminServer = new JokeAdminServer();

Thread t = new Thread(jokeAdminServer);

t.start();

/\* Check if user has input an argument, and if that argument is 'secondary'.

\* If so, start the secondary server in another thread. If there is an

\* argument and it is not 'secondary', an error message is sent and the

\* secondary doesn't start, but the primary stil does\*/

if(a.length == 1)

{

if(a[0].equals("secondary"))

{

SecondaryJokeServer secondaryJokeServer = new SecondaryJokeServer();

Thread thread = new Thread(secondaryJokeServer);

thread.start();

SecondaryAdminServer secondaryAdminServer = new SecondaryAdminServer();

Thread secAdminThread = new Thread(secondaryAdminServer);

secAdminThread.start();

}

else

{

System.out.println("Invalid parameter");

}

}

System.out.println("Joke server 1.8 starting....listening at port " + port);

while(controlSwitch)

{

socket = serverSocket.accept();

new Worker(socket).start();

}

}// End main

}//End class JokeServer

class JokeAdminServer implements Runnable

{

public static boolean adminControlSwitch = true;

// Running the JokeClientAdmin listen loop

public void run()

{

int q\_len = 6; // Maximum length of the queue for incoming connections. If this is exceeded, connection is refused

int port = 5050; // Admin client is listening at port 5050 at a different port for Admin clients

Socket sock; // A Socket object that will be used for communication with the client

try

{

/\*Create server socket that listens for requests from a client on the port hard-coded above.

\* The server socket allows up to 6 connections \*/

ServerSocket servsock = new ServerSocket(port, q\_len);

while (adminControlSwitch)

{

/\*The server socket will listen for a connection to the socket and accept it once a connection

has been made, creating a new socket to use for communication with the client. Program execution

is blocked until connection is made \*/

sock = servsock.accept();

/\*Start Worker thread to handle the actual processing of client request. The socket that

\* was created is passed to the Worker constructor. The Worker class extends the

\* Thread class, so it is a thread), \*/

new AdminWorker (sock).start();

}

servsock.close();

}catch (IOException ioe)

{

System.out.println(ioe);

}

}

}//End class AdminLooper

/\* This is the class for the JokeClientAdmin's worker thread.

\* The constructor takes in a socket, which is a connection to

\* the server on port 5050, and sets this socket to its local

\* variable, adminSocet.\*/

class AdminWorker extends Thread

{

Socket adminSocket;

String mode = "";

AdminWorker (Socket sock)

{

adminSocket = sock;

}

/\*The run() method is called when the thread is started and processes the client request.

\* It creates input/output streams, read in a mode from the user and passes this to

\* the server \*/

public void run()

{

BufferedReader inputStream = null;

PrintStream outputStream = null;

try

{

inputStream = new BufferedReader(new InputStreamReader(adminSocket.getInputStream())); // Stream to read from

outputStream = new PrintStream(adminSocket.getOutputStream()); // Stream to write to

outputStream.println(JokeServer.serverMode); // Send current mode to JokeAdminClient so it can be toggled

outputStream.flush();

String mode = inputStream.readLine(); // Read in the mode from the AdminClient user

System.out.println("Mode is: " + mode); // Print a message to the console on server

outputStream.println("Setting mode to " + mode); // Send message to JokeClientAdmin user that mode is being set

JokeServer.serverMode = mode; // Set the server mode to that which the user entered

adminSocket.close(); // Close the socket to prevent resource leak

}//End try

catch(IOException e)

{

System.out.println(e); //Exception creating the input/output streams

}

}

}// End class AdminWorker

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Code for SecondaryJokeServer \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class SecondaryWorker extends Thread

{

Socket secSocket;

String joke;

String[] jokes = new String[4];

String[] proverbs = new String[4];

static int jIndex;

static int pIndex;

ArrayList<String> jokePrefixList = new ArrayList<>();

ArrayList<String> proverbPrefixList = new ArrayList<>();

SecondaryWorker (Socket sock)

{

secSocket = sock;

initializeJokesAndProverbsLists();

initializeJokesAndProverbsPrefixLists();

}

/\* These ArrayLists are used to help keep track of the jokes/proverbs that have

\* already been sent to the client. The jokes/proverbs are saved in separate

\* hash tables where a UUID (provided by the user) is the key and the value is

\* a list of jokes/proverbs for that user. The approach taken was to loop

\* through the list of jokes/proverbs for a given user and pull off the prefixes

\* and store them in a list. Then, for every prefix in this list, use the

\* indexOf() method on the jokePrefixArray and proverbPrefixArray to

\* find the indexes for the jokes/proverbs that have already been used.

\*

\* For example, assume client was sent jokes B and C:

\*

\* Access the user's entry in the joke table using the

\* id sent by the client.

\*

\* Loop through the list of jokes for this client, pull off the

\* prefixes for jokes B and C. The list would look like => [JB, JC]

\*

\* Use the fact that the first joke in the joke array is prefixed

\* with JA, the second with JB, etc., you can use the indexOf

\* method on the jokePrefixArray to find the corresponding index

\* into the joke array. In this case, call to indexOf for JB would

\* give 1, and that means that jokes[1] has already been sent to

\* the client.

\*

\* A similar example would apply for proverbs.

\* \*/

private void initializeJokesAndProverbsPrefixLists()

{

jokePrefixList.add("JA");

jokePrefixList.add("JB");

jokePrefixList.add("JC");

jokePrefixList.add("JD");

proverbPrefixList.add("PA");

proverbPrefixList.add("PB");

proverbPrefixList.add("PC");

proverbPrefixList.add("PD");

}

/\* This method populates the lists that will hold jokes and proverbs,

\* in order. That is, the first entry holds joke A, the

\* second hold joke B, etc. Since you don't want to send

\* the same joke to the client until all 4 jokes are sent,

\* you need to keep track of which jokes are "valid" to send

\* to the user. This arrangement makes it simple to know

\* the valid indexes in this array once you learn which

\* joke(s) have already been sent to the client. For

\* example, if you know that joke A has been sent, then

\* you know that the first index is not valid.

\*

\* The "<#name>" is used as a placeholder so that the uer's

\* name can be inserted into the jokes and proverbs sent

\* from the server.\*/

private void initializeJokesAndProverbsLists()

{

jokes[0] = "JA <#name>: I saw a wino eating grapes.I told him, you gotta wait. (Mitch Hedberg)";

jokes[1] = "JB <#name>: I couldn’t believe that the highway department called my dad a thief. But when I got home, all the signs were there.";

jokes[2] = "JC <#name>: Why shouldn't you write with a broken pencil? Because it's pointless. ";

jokes[3] = "JD <#name>: What did 0 say to 8? Nice belt!";

proverbs[0] = "PA <#name>: Beauty is in the eye of the beholder";

proverbs[1] = "PB <#name>: Don’t bite the hand that feeds you";

proverbs[2] = "PC <#name>: Fortune favors the bold";

proverbs[3] = "PD <#name>: There is no time like the present";

}

public void run()

{

BufferedReader inputStream = null;

PrintStream outputStream = null;

try

{

/\*Create input and output streams to read data from and write data to the socket,

\* thus allowing communication between server and client.

\* The underlying input stream that is read from is the socket's input stream.

\* The underlying output stream that is written to is the socket's output stream \*/

inputStream = new BufferedReader(new InputStreamReader(secSocket.getInputStream()));

outputStream = new PrintStream(secSocket.getOutputStream());

try

{

// Get the user's name sent from the client and put it in a variable

String user = inputStream.readLine();

//Get the id sent from the user and put it in a variable

String id = inputStream.readLine();

/\* Default mode is joke-mode, and any string other than "proverb-mode" or

\* "quit" is taken to be "joke-mode\*/

if(SecondaryJokeServer.serverMode.equals("proverb-mode"))

{

sendMessage(id, user, outputStream, proverbs, SecondaryJokeServer.proverbMap, proverbPrefixList);

}

else //Server is in joke-mode

{

sendMessage(id, user, outputStream, jokes, SecondaryJokeServer.jokeMap, jokePrefixList);

}

}//End inner try

catch(IOException ex)

{

System.out.println("Server read error"); //Exception trying to read from input stream or write to output stream

ex.printStackTrace();

}

secSocket.close();

}//End outer try

catch(IOException e)

{

System.out.println(e); //Exception creating the input/output streams

}

}

/\* This method takes the user's name, user id, the output stream for the server, an array

\* that contains jokes or proverbs, and a hash map that consists of the jokes or proverbs

\* that were sent to the user with the given id

\*

\* The method finds a joke or proverb (depending on the hash map) that has not yet been

\* sent to the user and sends it.

\*/

public void sendMessage(String id, String user, PrintStream out, String[] messageArray, Map<String, ArrayList<String>> map, ArrayList<String> prefixList)

{

String messageFromServer = "";

/\* Holds the indexes for jokes[] or proverbs[] array for the jokes or proverbs that have

\* already been sent to the client \*/

ArrayList<Integer> existingIndexes = new ArrayList<>();

// Check if the user has connected before

if(map.get(id) == null)

{

/\* If not, create an entry for the user in the appropriate hash map

\* with the user's id as the key and an empty list \*/

map.put(id, new ArrayList<String>());

/\* Since the user has not connected and he/she has not received any

\* thing from the server as of yet. This means that any joke or

\* proverb is valid. The message array contains 4 items, so get a

\* number from 0 to 3 and get the joke or proverb in the joke or proverb array

\* at that index. Then, send it to the client on the output stream.\*/

Random pRandom = new Random();

int pIndex = pRandom.nextInt(4);

messageFromServer = messageArray[pIndex];

messageFromServer = messageFromServer.replaceAll("<#name>", user);

out.println("<S2> " + messageFromServer);

//Add the joke or proverb to the user's list of jokes or proverbs that have been sent

map.get(id).add(messageFromServer);

//printMap(map, user);

}

else //The else statement is hit when the user has connected previously

{

/\* Get the jokes or proverbs that have already been sent to this user by

looking them up in the appropriate hash map\*/

ArrayList<String> msgs = map.get(id);

/\* Get the indexes into to the joke or proverb array for the

\* jokes or proverbs that have already been sent to the user \*/

existingIndexes = getIndexesOfPriorMessages(id, map, prefixList);

/\* Get a random number between 0 and 3 and if that the

\* generated number is in the list of indexes that

\* represent jokes/proverbs already sent to the user, keep getting

\* a random number until you get one that is not in the list of

\* indexes for jokes or proverbs already sent to the client\*/

Random r1 = new Random();

int randomIndex1 = r1.nextInt(4);

while(existingIndexes.contains(randomIndex1))

{

randomIndex1 = r1.nextInt(4);

}

/\* Once an index if found, use it to index into the jokes or proverbs

\* array to find a joke or proverb that has not yet been sent to the

\* client, insert the user name into the joke or proverb and send the

\* joke or proverb to the client.

\*

\* Lastly, add that joke or proverb to the list of jokes or proverbs for

\* that user id \*/

messageFromServer = messageArray[randomIndex1];

messageFromServer = messageFromServer.replaceAll("<#name>", user);

out.println("<S2> " + messageFromServer);

msgs.add(messageFromServer);

//printMap(map, user);

if(msgs.size() == 4)

{

msgs.clear();

}

}

}// End sendMessage()

/\* This method takes the user id, a hash map of jokes or proverbs, and

\* a list prefixes for jokes or provers, i.e., [JA, JC] or [PA, PD]

\*

\* This method returns a list of integers that are indexes of the jokes

\* or proverb array for the jokes or proverbs that have already been sent

\* to the user

\*

\* It first gets the list of jokes or proverbs (depending on the hashmap that

\* has been passed in) that have already been sent to this user. This is

\* accomplished by doing a lookup of the hash table, using the passed-in id as

\* the key.

\*

\* Then, for each joke in the list, it pulls off the the prefix,

\* for example, JA, JC, and JD and uses indexOf() on the jokePrefixArray

\* to find the corresponding index for each prefix.

\*

\* For the example above, for JA, JC, and JD, the method would return

\* the list [0, 2, 3]

\*/

ArrayList<Integer> getIndexesOfPriorMessages(String id, Map<String, ArrayList<String>> map, ArrayList<String> prefixes)

{

ArrayList<String> msgs = map.get(id);

ArrayList<Integer> mIndexes = new ArrayList<>();

for(String s : msgs)

{

String prefix = s.substring(0, 2);

mIndexes.add(prefixes.indexOf(prefix));

}

return mIndexes;

}// End getIndexesOfPriorMessages()

//Print out the joke or proverb hash map to verify things are working as expected

void printMap(Map<String, ArrayList<String>> m, String uName)

{

System.out.println("\nMap for: " + uName);

for(String k:m.keySet())

{

ArrayList<String> value = m.get(k);

System.out.print(k + ":");

for(String s:value)

{

s = s.substring(0, 2);

System.out.print(s + "\t");

}

}

}// End printMap()

} //End class SecondaryWorker

class SecondaryJokeServer implements Runnable

{

public static boolean secControlSwitch = true;

public static String serverMode = "joke-mode"; // Default mode is joke-mode

/\* Hash maps to store the jokes and proverbs for each client (user). The hash maps

\* use the UUID sent from the client as a key and the value is the list of jokes

\* or proverbs that have been sent to that client \*/

public static Map<String, ArrayList<String>> jokeMap = new HashMap<>();

public static Map<String, ArrayList<String>> proverbMap = new HashMap<>();

public void run()

{

int qSec\_len = 6; // Maximum length of the queue for incoming connections. If this is exceeded, connection is refused

int secPort = 4546; // Secondary server is listening at port 5050 at a different port for Admin clients

Socket secSock; // A Socket object that will be used for communication with the client

try

{

/\*Create server socket that listens for requests from a client on the port hard-coded above.

\* The server socket allows up to 6 connections \*/

ServerSocket secServsock = new ServerSocket(secPort, qSec\_len);

System.out.println("Joke server 1.8 starting....listening at port " + secPort);

while (secControlSwitch)

{

/\*The server socket will listen for a connection to the socket and accept it once a connection

has been made, creating a new socket to use for communication with the client. Program execution

is blocked until connection is made \*/

secSock = secServsock.accept();

/\*Start Worker thread to handle the actual processing of client request. The socket that

\* was created is passed to the Worker constructor. The Worker class extends the

\* Thread class, so it is a thread), \*/

new SecondaryWorker(secSock).start();

}

secServsock.close();

}

catch (IOException ioe)

{

System.out.println(ioe);

}

}// End run()

}// End class SecondaryJokeServer

class SecondaryAdminServer implements Runnable

{

public static boolean secAdminControlSwitch = true;

public void run()

{

int q\_len = 6; // Maximum length of the queue for incoming connections. If this is exceeded, connection is refused

int port = 5051; // Admin client is listening at port 5050 at a different port for Admin clients

Socket sock; // A Socket object that will be used for communication with the client

try

{

/\*Create server socket that listens for requests from a client on the port hard-coded above.

\* The server socket allows up to 6 connections \*/

ServerSocket servsock = new ServerSocket(port, q\_len);

while (secAdminControlSwitch)

{

/\*The server socket will listen for a connection to the socket and accept it once a connection

has been made, creating a new socket to use for communication with the client. Program execution

is blocked until connection is made \*/

sock = servsock.accept();

/\*Start Worker thread to handle the actual processing of client request. The socket that

\* was created is passed to the Worker constructor. The Worker class extends the

\* Thread class, so it is a thread), \*/

new SecondaryAdminWorker (sock).start();

}

servsock.close();

}catch (IOException ioe)

{

System.out.println(ioe);

}

}

}// End class SecondaryAdminServer

class SecondaryAdminWorker extends Thread

{

Socket secAdminSocket;

String mode = "";

SecondaryAdminWorker (Socket sock)

{

secAdminSocket = sock;

}

/\*The run() method is called when the thread is started and processes the client request.

\* It creates input/output streams, read in a mode from the user and passes this to

\* the server \*/

public void run()

{

BufferedReader inputStream = null;

PrintStream outputStream = null;

try

{

inputStream = new BufferedReader(new InputStreamReader(secAdminSocket.getInputStream())); // Stream to read from

outputStream = new PrintStream(secAdminSocket.getOutputStream()); // Stream to write to

outputStream.println(SecondaryJokeServer.serverMode); // Send current mode to JokeAdminClient so it can be toggled

outputStream.flush();

String mode = inputStream.readLine(); // Read in the mode from the AdminClient user

System.out.println("<S2> Mode is: " + mode); // Print a message to the console on server

outputStream.println("<S2> Setting mode to " + mode); // Send message to AdminClient user that mode is being set

SecondaryJokeServer.serverMode = mode; // Set the server mode to that which the user entered

secAdminSocket.close(); // Close the socket to prevent resource leak

}//End try

catch(IOException e)

{

System.out.println(e); //Exception creating the input/output streams

}

}

} // End class SecondaryAdminWorker

User only connected to primary server only type ísí to toggle to secondary

and gets a message that no secondary is available. Note that output from

user Bobís session was captured after parts A, B, C were completed so his

session is in proverb mode. (Part C does not affect this user since he is

only connected to the primary server)

> java JokeClient

Starting Joke Client, 1.8

Using server: localhost, Port: 4545

Enter your name: Bob

s

No secondary server, using localhost on 4545

PB Bob: Early to bed, early to rise

quit

Goodbye, Bob!

Part A. Output shows client connected to primary and secondary server with 8 output responses in

joke mode and 8 responses in proverb mode. The first four jokes sent were JD, JC, JB, JA are

sent and now that all four have been used, they may be reused. The next four that are sent

are JA, JB, JC, JD.

The JokeClientAdmin switches to proverb mode and four proverbs are sent. PB, PA, PC, PD and

since all four have been sent, they may be reused. The next four sent are PD, PB, PC, PA.

B. User then enters ísí and is toggled to the secondary server, and the JokeClientAdmin

changes to secondary as well.

C. Repeat part A for the secondary server

ó Output from JokeClientAdmin for this session:

> java JokeClientAdmin localhost localhost

Starting Joke Admin Client, 1.8

Primary server: localhost, Port: 5050

Secondary server: localhost, Port: 5051

Setting mode to proverb-mode <== Changed to proverb mode for part A on primary

s

Using SECONDARY server localhost on port 5051 <== Toggled to secondary server

<S2> Setting mode to proverb-mode <== Changed to proverb mode for part C on secondary

quit

Goodbye!

ó- Output on server for this session

> java JokeServer secondary

Joke server 1.8 starting....listening at port 4546

Joke server 1.8 starting....listening at port 4545

Mode is: proverb-mode

<S2> Mode is: proverb-mode

> java JokeClient localhost localhost

Starting Joke Client, 1.8

Using server: localhost, Port: 4545

Using server: localhost, Port: 4546

Enter your name: Alice

JD Alice: I had a dream that I was a muffler last night. I woke up exhausted! <== Begin A

JC Alice: Why did the Clydesdale give the pony a glass of water? Because he was a little horse!

JB Alice: What time did the man go to the dentist? Tooth hurt-y.î

JA Alice: A recent scientific study showed that out of 2,293,618,367 people, 94% are too lazy to actually read that number.

JA Alice: A recent scientific study showed that out of 2,293,618,367 people, 94% are too lazy to actually read that number.

JB Alice: What time did the man go to the dentist? Tooth hurt-y.î

JC Alice: Why did the Clydesdale give the pony a glass of water? Because he was a little horse!

JD Alice: I had a dream that I was a muffler last night. I woke up exhausted!

PB Alice: Early to bed, early to rise <== Begin 8 proverbs

PA Alice: There is no such thing as a free lunch

PC Alice: A picture is worth a thousand words

PD Alice: Absence makes the heart grow fonder

PD Alice: Absence makes the heart grow fonder

PB Alice: Early to bed, early to rise

PC Alice: A picture is worth a thousand words

PA Alice: There is no such thing as a free lunch <== End A

s <== SWITCH TO SECONDARY, PART B

Using localhost on port 4546

<S2> JB Alice: I couldnít believe that the highway department called my dad a thief. But when I got home, all the signs were there. <= C

<S2> JC Alice: Why shouldn't you write with a broken pencil? Because it's pointless.

<S2> JD Alice: What did 0 say to 8? Nice belt!

<S2> JA Alice: I saw a wino eating grapes.I told him, you gotta wait. (Mitch Hedberg)

<S2> JB Alice: I couldnít believe that the highway department called my dad a thief. But when I got home, all the signs were there.

<S2> JD Alice: What did 0 say to 8? Nice belt!

<S2> JA Alice: I saw a wino eating grapes.I told him, you gotta wait. (Mitch Hedberg)

<S2> JC Alice: Why shouldn't you write with a broken pencil? Because it's pointless.

<S2> PC Alice: Fortune favors the bold

<S2> PD Alice: There is no time like the present

<S2> PB Alice: Donít bite the hand that feeds you

<S2> PA Alice: Beauty is in the eye of the beholder

<S2> PA Alice: Beauty is in the eye of the beholder

<S2> PC Alice: Fortune favors the bold

<S2> PB Alice: Donít bite the hand that feeds you

<S2> PD Alice: There is no time like the present <== End C

quit

Goodbye, Alice!