**AsyncJokeServer.java**

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

1. Name / Date: Vatsal Parikh, May 28, 2022

2. Java version used (java -version), if not the official version for the class:

java 18.0.1.1

3. Precise command-line compilation examples / instructions:

> javac AsyncJokeServer.java

> javac AsyncJokeClient.java

> javac AsyncJokeAdminClient.java

4. Precise examples / instructions to run this program:

In separate shell windows:

> java AsyncJokeServer

> java AsyncJokeClient

> java AsyncJokeAdminClient

All acceptable commands are displayed below:

> java AsyncJokeServer secondary

> java AsyncJokeClient

> java AsyncJokeClient localhost

> java AsyncJokeClient localhost localhost

> java AsyncJokeClient 127.0.0.1 localhost

> java AsyncJokeAdminClient

> java AsyncJokeAdminClient localhost

> java AsyncJokeAdminClient localhost localhost

5. List of files needed for running the program.

a. AsyncJokeServer.java

b. AsyncJokeClient.java

c. AsyncJokeAdminClient.java

5. Notes:

It is a multithreaded UDP server/client with a secondary server for both Client and Admin.

I didn't transfer objects between servers and clients instead I stored only UUID

at client side and passed it as string between server and client adding more security

but also complexity at server side.

I used hashtables for storing state as it is easier than custom classes to iterate over.

Only the shuffle of Joke IDs are stored as state for users and not all jokes/proverbs.

There is only one copy of jokes/proverbs.

In the worker thread, once a request has been received, sleeping for 40 seconds.

After sending the asynchronous request for Joke/Proverb the client displays a prompt

on the client console that asks the user to enter two numbers and returning sum to show

work on server side.

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

// Importing Java Input Output and networking libraries

import java.io.\*;

import java.net.\*;

import java.util.\*;

// This will act as the parent class whose variable can be used in both AdminWorker and ServerWorker

class Worker extends Thread{

// mode\_admin and mode\_client will be used keep track of the current mode and both servers

static Hashtable<Integer, String> mode\_admin = new Hashtable<Integer, String>();

static Hashtable<Integer, String> mode\_client = new Hashtable<Integer, String>();

}

// AdminWorker class

class AdminWorker extends Worker {

ServerSocket servsock;

Socket sock;

// Passsing the serversocket instead of socket as argument in constructor

AdminWorker(ServerSocket servsock) {

this.servsock = servsock;

// Inserting default values

mode\_admin.putIfAbsent(5050, "joke");

mode\_admin.putIfAbsent(5051, "joke");

mode\_client.putIfAbsent(4545, "joke");

mode\_client.putIfAbsent(4546, "joke");

}

// start() method

public void run() {

while(true){

try{

// Connecting to admin client

sock = servsock.accept();

int admin\_port = servsock.getLocalPort();

PrintStream out = null;

BufferedReader in = null;

// Trying to assign in/out to socket's(client) Input/Output stream and returning error if not possible

try {

in = new BufferedReader(new InputStreamReader(sock.getInputStream()));

out = new PrintStream(sock.getOutputStream());

// Trying to read client request and if it cannot be read returning an error

try {

// Sending the current mode to admin

out.println(mode\_admin.get(admin\_port));

// Reading the mode returned by admin, changing the old value in mode\_admin and printing it on server side

mode\_admin.replace(admin\_port, in.readLine());

System.out.println("Changed mode for " + admin\_port +" to " + mode\_admin.get(admin\_port));

// For the current port when mode if changed it will be reflected on admin\_port for server worker to use

if(admin\_port==5050){

mode\_client.replace(4545, mode\_admin.get(admin\_port));

}

else if(admin\_port==5051){

mode\_client.replace(4546, mode\_admin.get(admin\_port));

}

} catch (Exception x) {}

// Closing the connection with client, though server is still running

sock.close();

} catch (IOException ioe) {System.out.println(ioe);}

} catch(Exception e){

System.out.println(e);

}

}

}

static String toText (byte ip[]) {

StringBuffer result = new StringBuffer ();

for (int i = 0; i < ip.length; ++ i) {

if (i > 0) result.append (".");

result.append (0xff & ip[i]);

}

return result.toString ();

}

}

// ServerWorker class

class ServerWorker extends Worker {

// buffer will contain the response received by client

byte[] buffer;

// request and ds contains client information

DatagramPacket request;

DatagramSocket ds;

// Defining jokes and proverbs

static Hashtable<String, String> jokes = new Hashtable<String, String>();

static Hashtable<String, String> proverbs = new Hashtable<String, String>();

// According to server used and user accessing the server, user's joke/proverb sequence will be stored here

static Hashtable<Integer, Hashtable<String, ArrayList<String>>> server\_jokes = new Hashtable<Integer, Hashtable<String, ArrayList<String>>>();

static Hashtable<Integer, Hashtable<String, ArrayList<String>>> server\_proverbs = new Hashtable<Integer, Hashtable<String, ArrayList<String>>>();

// This will contain the shuffle used to randomize the joke sequence

ArrayList<String> shuffle\_jokes;

ArrayList<String> shuffle\_proverbs;

static String out = "";

// Worker Constructor

ServerWorker (DatagramSocket ds, DatagramPacket request, byte[] buffer) {

// Putting jokes and proverbs in static jokes and proverbs hashtable with respective keys

jokes.put("JA", "A clean house is the sign of a broken computer.");

jokes.put("JB", "CAPS LOCK has been preventing login since 1980.");

jokes.put("JC", "What do you call 8 hobbits? A hobbyte");

jokes.put("JD", "I love the F5 key. It is so refreshing.");

proverbs.put("PA", "I do not fear computers. I fear lack of them.");

proverbs.put("PB", "The computer was born to solve problems that did not exist before.");

proverbs.put("PC", "Computing is not about computers any more. It is about living.");

proverbs.put("PD", "Man is still the most extraordinary computer of all.");

mode\_client.putIfAbsent(4545, "joke");

mode\_client.putIfAbsent(4546, "joke");

// initializing ds, buffer and request variables

this.ds = ds;

this.buffer = buffer;

this.request = request;

}

// Run method will run when start() is called

public void run(){

// response variable contain the DatagramPacket

DatagramPacket response = null;

// Converting byte[] to String

String concat = (new String(buffer));

// Splitting and formatting the string to get new\_name, uuid and port

String new\_name = concat.split(",")[0];

String uuid = concat.split(",")[1];

int port = Integer.parseInt(concat.split(",")[2].substring(0,4));

// Retrieving clietn\_port from request variable

int client\_port = request.getPort();

server\_jokes.putIfAbsent(port, new Hashtable<>());

server\_proverbs.putIfAbsent(port, new Hashtable<>());

// Trying to send the Packet and returning error if not possible

try {

// Simulating work by sleeping the worker thread

Thread.sleep(40000);

// Requesting address

InetAddress address = request.getAddress();

// Checking if the mode is joke and adding the shuffled list if empty to respective user to respective server

if(mode\_client.get(port).equals("joke")){

shuffle\_jokes = Collections.list(jokes.keys());

Collections.shuffle(shuffle\_jokes);

server\_jokes.get(port).putIfAbsent(uuid, shuffle\_jokes);

if(server\_jokes.get(port).get(uuid).isEmpty()){

// When the shuffled list is empty message is sent to client

if(port==4546) out = "<S2>";

out += "\nJOKE CYCLE COMPLETED\n";

server\_jokes.get(port).put(uuid, shuffle\_jokes);

}

// Calling the printstate function

printState(new\_name, port, server\_jokes.get(port).get(uuid).remove(0));

// Creating the Datagram packet to sent to client

response = new DatagramPacket(out.getBytes(), out.getBytes().length, address, client\_port);

// Sending the response

ds.send(response);

out = "";

}

// Checking if the mode is joke and adding the shuffled list if empty to respective user to respective server

else if(mode\_client.get(port).equals("proverb")){

shuffle\_proverbs = Collections.list(proverbs.keys());

Collections.shuffle(shuffle\_proverbs);

server\_proverbs.get(port).putIfAbsent(uuid, shuffle\_proverbs);

if(server\_proverbs.get(port).get(uuid).isEmpty()){

// When the shuffled list is empty message is sent to client

if(port==4546) out = "<S2>";

out += "\nPROVERB CYCLE COMPLETED\n";

server\_proverbs.get(port).put(uuid, shuffle\_proverbs);

}

// Calling the printstate function

printState(new\_name, port, server\_proverbs.get(port).get(uuid).remove(0));

// Creating the Datagram packet to sent to client

response = new DatagramPacket(out.getBytes(), out.getBytes().length, address, client\_port);

// Sending the response

ds.send(response);

out = "";

}

// Catching the exception

} catch (Exception ioe) {System.out.println(ioe);}

}

// This static method will update the out variable which will be later sent to client

static void printState(String name, int client\_port, String shuffle) {

try {

// When second server is connected <S2> is added

if(client\_port==4546) out = "<S2> ";

out += shuffle + " " + name + ": ";

if(mode\_client.get(client\_port).equals("joke")){

out += jokes.get(shuffle);

}

else if(mode\_client.get(client\_port).equals("proverb")){

out += proverbs.get(shuffle);

}

} catch(Exception ex) {

out += "Failed!" + name;

}

}

}

// Serverlooper class

class ServerLooper implements Runnable {

// If this boolean value is true we can switch the server

public static boolean ServerSwitch = true;

// RuNning the Admin listen loop

public void run(){

int q\_len = 6;

// buffer will contain the response received by client

byte[] buffer = new byte[100];

// request contains client information

DatagramPacket request = null;

try{

DatagramSocket ds = new DatagramSocket(4546);

ServerSocket servsock\_admin = new ServerSocket(5051, q\_len);

new AdminWorker(servsock\_admin).start();

while (ServerSwitch) {

// wait for the next ADMIN client connection:

request = new DatagramPacket(buffer, buffer.length);

ds.receive(request);

new ServerWorker(ds, request, buffer).start();

}

}catch (Exception e){

System.out.println(e);

}

}

}

// AsyncJokeServer class

public class AsyncJokeServer {

// Main method

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

// Socket used to send UDP packet

DatagramSocket ds = new DatagramSocket(4545);

// buffer will contain the response received by client

byte[] buffer = new byte[100];

int q\_len = 6;

System.out.println("Vatsal Parikh's Async Joke Server is starting up, listening at port 4545.\n");

DatagramPacket request = null;

// If the argument "secondary" is added then Serverlooper is called

if(args.length>0){

if(args[0].contains("secondary")){

System.out.println("Vatsal Parikh's Async Joke Server 2 is starting up, listening at port 4546.\n");

ServerLooper SL = new ServerLooper();

Thread t = new Thread(SL);

t.start();

}

}

// Calling AdminWorker with server socket

ServerSocket servsock\_admin = new ServerSocket(5050, q\_len);

new AdminWorker(servsock\_admin).start();

while(true){

// Accepting requests from clients

request = new DatagramPacket(buffer, buffer.length);

ds.receive(request);

new ServerWorker(ds, request, buffer).start();

}

}

}

**AsyncJokeClient.java**

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

1. Name / Date: Vatsal Parikh, May 28, 2022

2. Java version used (java -version), if not the official version for the class:

java 18.0.1.1

3. Precise command-line compilation examples / instructions:

> javac AsyncJokeServer.java

> javac AsyncJokeClient.java

> javac AsyncJokeAdminClient.java

4. Precise examples / instructions to run this program:

In separate shell windows:

> java AsyncJokeServer

> java AsyncJokeClient

> java AsyncJokeAdminClient

All acceptable commands are displayed below:

> java AsyncJokeServer secondary

> java AsyncJokeClient

> java AsyncJokeClient localhost

> java AsyncJokeClient localhost localhost

> java AsyncJokeClient 127.0.0.1 localhost

> java AsyncJokeAdminClient

> java AsyncJokeAdminClient localhost

> java AsyncJokeAdminClient localhost localhost

5. List of files needed for running the program.

a. AsyncJokeServer.java

b. AsyncJokeClient.java

c. AsyncJokeAdminClient.java

5. Notes:

It is a multithreaded UDP server/client with a secondary server for both Client and Admin.

I didn't transfer objects between servers and clients instead I stored only UUID

at client side and passed it as string between server and client adding more security

but also complexity at server side.

I used hashtables for storing state as it is easier than custom classes to iterate over.

Only the shuffle of Joke IDs are stored as state for users and not all jokes/proverbs.

There is only one copy of jokes/proverbs.

In the worker thread, once a request has been received, sleeping for 40 seconds.

After sending the asynchronous request for Joke/Proverb the client displays a prompt

on the client console that asks the user to enter two numbers and returning sum to show

work on server side.

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

// Importing Java Input Output and networking libraries

import java.io.\*;

import java.net.\*;

import java.util.\*;

// AsyncJokeClient class

public class AsyncJokeClient{

// UUID with respective name is stored in name\_uuid

static Hashtable<String, String> name\_uuid = new Hashtable<String, String>();

static String name;

static String textFromServer = null;

// Main method

public static void main (String args[]) {

String serverName1 = "localhost";

String serverName2 = "localhost";

String currentServer = serverName1;

int port\_server1 = 4545;

int port\_server2 = 4546;

int currentPort = port\_server1;

// Switcher will be used to check if user provided multiple arguments(ie multi server)

boolean switcher = false;

// Based on arguments servername will be fetched

if(args.length == 1) {

serverName1 = args[0];

}

System.out.println("Vatsal Parikh's Async Joke Client\n");

System.out.println("Server one: " + serverName1 + ", Port: 4545");

// If second hostname is given then option is given to change servers

if (args.length == 2){

serverName2 = args[1];

System.out.println("Server two: " + serverName1 + ", Port: 4546");

System.out.println("\n\*\*(s) to change servers\*\*");

switcher = true;

}

// Taking input from the user using InputStreamReader

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

try {

do {

// Fetching user input

System.out.print("\nEnter your name or just press enter(if you are the previous user), \n(quit) to end: ");

System.out.flush ();

String temp = in.readLine();

// Changing servers when the input is s and printing accordingly on admin side

if(temp.equals("s")){

// Changing boolean value

if(switcher){

if(currentPort==port\_server1){

currentServer = serverName2;

System.out.println("Now communicating with: localhost, port 4546");

currentPort = port\_server2;

}

else if(currentPort==port\_server2){

currentServer = serverName1;

System.out.println("Now communicating with: localhost, port 4545");

currentPort = port\_server1;

}

continue;

}

// If no secondary server is provided it will provide this output

else{

System.out.println("No secondary server being used");

continue;

}

}

if(!temp.isEmpty()){

name = temp;

}

// If user input is valid, getJoke method is called with user input and server name

if (name.indexOf("quit") < 0){

if(!name\_uuid.keySet().contains(name)){

name\_uuid.put(name, UUID.randomUUID().toString());

}

getJoke(name, name\_uuid.get(name), currentServer, currentPort);

}

textFromServer = null;

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

System.out.println ("Cancelled by user request.");

} catch (Exception x) {System.out.println(x);}

}

// This static method will send the name, uuid and currentPort to server and shpwing some work done

static void getJoke (String name, String uuid, String serverName, int currentPort) throws Exception{

// Defining InetAddress, socket and toServer

InetAddress address = InetAddress.getByName(serverName);

DatagramSocket socket = new DatagramSocket();

byte[] toServer = null;

// Trying to send the request to user and returning error if not possible

try {

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

// Concatenating the reponse text

String temp = name + "," + uuid + "," + currentPort;

// Converting String to byte[]

toServer = temp.getBytes();

// Connecting to server with default port

DatagramPacket request = new DatagramPacket(toServer, toServer.length, address, 4545);

// Sending the request to server

socket.send(request);

// Starting a thread to listen for Server response

new ServerResponse(socket).start();

// While we receive the response simulating work using basic sum calculating

while(textFromServer==null){

try{

System.out.print("\nEnter numbers seperated by space to sum: ");

String[] nums = in.readLine().split(" ");

int num1 = Integer.parseInt(nums[0]);

int num2 = Integer.parseInt(nums[1]);

System.out.println("Your sum is: " + (num1 + num2));

}

catch(Exception e){

System.out.println("Try again!");

}

}

// Printing out the response

System.out.println(textFromServer);

} catch (IOException x) {

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

x.printStackTrace ();

}

}

// ServerReponse class

static class ServerResponse extends Worker{

DatagramSocket socket;

byte[] fromServer = new byte[100];

DatagramPacket response;

// Intializing socket

ServerResponse(DatagramSocket socket){

this.socket = socket;

}

// start() method

public void run(){

try{

// Waiting for the server response

response = new DatagramPacket(fromServer, fromServer.length);

socket.receive(response);

textFromServer = new String(fromServer, 0, response.getLength());

}

catch(Exception e){System.out.println(e);}

}

}

}

**AsyncJokeAdminClient.java**

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

1. Name / Date: Vatsal Parikh, May 28, 2022

2. Java version used (java -version), if not the official version for the class:

java 18.0.1.1

3. Precise command-line compilation examples / instructions:

> javac AsyncJokeServer.java

> javac AsyncJokeClient.java

> javac AsyncJokeAdminClient.java

4. Precise examples / instructions to run this program:

In separate shell windows:

> java AsyncJokeServer

> java AsyncJokeClient

> java AsyncJokeAdminClient

All acceptable commands are displayed below:

> java AsyncJokeServer secondary

> java AsyncJokeClient

> java AsyncJokeClient localhost

> java AsyncJokeClient localhost localhost

> java AsyncJokeClient 127.0.0.1 localhost

> java AsyncJokeAdminClient

> java AsyncJokeAdminClient localhost

> java AsyncJokeAdminClient localhost localhost

5. List of files needed for running the program.

a. AsyncJokeServer.java

b. AsyncJokeClient.java

c. AsyncJokeAdminClient.java

5. Notes:

It is a multithreaded UDP server/client with a secondary server for both Client and Admin.

I didn't transfer objects between servers and clients instead I stored only UUID

at client side and passed it as string between server and client adding more security

but also complexity at server side.

I used hashtables for storing state as it is easier than custom classes to iterate over.

Only the shuffle of Joke IDs are stored as state for users and not all jokes/proverbs.

There is only one copy of jokes/proverbs.

In the worker thread, once a request has been received, sleeping for 40 seconds.

After sending the asynchronous request for Joke/Proverb the client displays a prompt

on the client console that asks the user to enter two numbers and returning sum to show

work on server side.

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

// Importing Java Input Output and networking libraries

import java.io.\*;

import java.net.\*;

// AsyncJokeAdminClient class

public class AsyncJokeAdminClient{

// Main method

public static void main (String args[]) {

// Defining servernames and port numbers

String serverName1 = "localhost";

String serverName2 = "localhost";

String currentServer = serverName1;

int port\_server1 = 5050;

int port\_server2 = 5051;

int currentPort = port\_server1;

// Switcher will be used to check if user provided multiple arguments(ie multi server)

boolean switcher = false;

// Based on arguments servername will be fetched

if(args.length == 1) {

serverName1 = args[0];

}

System.out.println("Vatsal Parikh's Admin Client\n");

System.out.println("Server one: " + serverName1 + ", Port: 5050");

if (args.length == 2){

serverName2 = args[1];

System.out.println("Server two: " + serverName1 + ", Port: 5051");

System.out.println("\n\*\*(s) to change servers\*\*\n");

switcher = true;

}

try {

// Defining I/O streams fromServer/toServer and in

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

BufferedReader fromServer;

PrintStream toServer;

// Trying to get user inputs until user passes quit as input

do {

Socket sock;

System.out.println("Press enter to change modes");

// Waiting for user input and it won't execute further until some kind of input is given

String temp = in.readLine();

// Trying to make a connection with the server name and port number

try{

sock = new Socket(currentServer, currentPort);

// Assigning fromServer/toServer to socket's(server) Input/Output stream

fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));

toServer = new PrintStream(sock.getOutputStream());

String mode="joke";

// If the input is quit and breaking the loop and exiting the program

if(temp.toString().contains("quit")){

break;

}

// Changing servers when the input is s and printing accordingly on admin side

if(temp.equals("s")){

// Changing boolean value

if(switcher){

if(currentPort==port\_server1){

currentServer = serverName2;

System.out.println("Now communicating with: localhost, port 5051\n");

currentPort = port\_server2;

}

else if(currentPort==port\_server2){

currentServer = serverName1;

System.out.println("Now communicating with: localhost, port 5050\n");

currentPort = port\_server1;

}

continue;

}

// If no secondary server is provided it will provide this output

else{

System.out.println("No secondary server being used");

continue;

}

}

// Server sends the current mode to admin

mode = fromServer.readLine();

// Based on the current mode admin will change the mode and send that that to server

if (mode.equals("joke")){

mode = "proverb";

}

else if(mode.equals("proverb")){

mode = "joke";

}

toServer.println(mode);

toServer.flush();

// Closing socket

sock.close();

} catch (Exception x) {

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

x.printStackTrace ();

}

} while(true);

} catch (IOException x) {x.printStackTrace();}

}

}

**Output:**

##AsyncJokeServer##

D:\Study\DS\Async Joke>java AsyncJokeServer secondary

Vatsal Parikh's Async Joke Server is starting up, listening at port 4545.

Vatsal Parikh's Async Joke Server 2 is starting up, listening at port 4546.

Changed mode for 5050 to proverb

Changed mode for 5051 to proverb

##AsyncJokeClient##

D:\Study\DS\Async Joke>java AsyncJokeClient localhost localhost

Vatsal Parikh's Async Joke Client

Server one: localhost, Port: 4545

Server two: localhost, Port: 4546

\*\*(s) to change servers\*\*

Enter your name or just press enter(if you are the previous user),

(quit) to end: vatsal

Enter numbers seperated by space to sum: 2 5

Your sum is: 7

Enter numbers seperated by space to sum: 4 6

Your sum is: 10

JA vatsal: A clean house is the sign of a broken computer.

Enter your name or just press enter(if you are the previous user),

(quit) to end: s

Now communicating with: localhost, port 4546

Enter your name or just press enter(if you are the previous user),

(quit) to end: vatsal

Enter numbers seperated by space to sum: 6 8

Your sum is: 14

Enter numbers seperated by space to sum: 1 3

Your sum is: 4

Enter numbers seperated by space to sum: 8 9

Your sum is: 17

<S2> PD vatsal: Man is still the most extraordinary computer of all.

Enter your name or just press enter(if you are the previous user),

(quit) to end:

##AsyncJokeAdminClient##

D:\Study\DS\Async Joke>java AsyncJokeAdminClient localhost localhost

Vatsal Parikh's Admin Client

Server one: localhost, Port: 5050

Server two: localhost, Port: 5051

\*\*(s) to change servers\*\*

Press enter to change modes

Press enter to change modes

s

Now communicating with: localhost, port 5051

Press enter to change modes

Press enter to change modes