/\*\*

\*Athina Verroiopoulou

\*1/9/2017

\*CSC435, InetServer

\*/

import java.net.ServerSocket;

import java.net.Socket;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.InetAddress;

import java.net.UnknownHostException;

/\*\*

\*

\* With the use of a thread we can have multiple processes that are executed at the same time in one (beginning) process

\* threads share the same space to save the address, so they can access the same object in a program

\*/

class Worker extends Thread {

Socket sock; //Create a local Socket value "sock"

//create the socket for communication, act as a constructor

Worker(Socket s) {

sock = s; //pass s as a socket parameter and assign it to sock

}

public void run() {

PrintStream out = null; //initialize the out value

BufferedReader in = null; //initialize the out value

try {

/\*\*

\* BufferedReader is used to read text from a character input stream, it buffers characters.

\* In Java Sockets every program can read and write in a Socket with the use of Input and OutputStream

\* that is a similar procedure with files. \*/

in = new BufferedReader(new InputStreamReader(sock.getInputStream())); //read from client , connect with socket

out = new PrintStream(sock.getOutputStream()); //write/respond to the client, connect with socket , for messages

try {

String name;//save the client response

name = in.readLine(); //read from the client the hostname or IP that is send with

System.out.println("Looking up " + name); //message with the hostname or IP

printRemoteAddress(name, out); //call the method printRemoteAddress and pass the given name and the Output object

} catch (IOException x) {

System.out.println("Server read error");//in case of error in connection

x.printStackTrace();

}

sock.close(); //we call the close method of Java.net.Socket class to terminate the connection with the specific client, not the server

} catch (IOException ioe) {//in case of error in reading

System.out.println(ioe);

}

}

//send remote address to client

static void printRemoteAddress(String name, PrintStream out) {//take the parameters for hostname or IP and output obj

try {

out.println("Looking up: " + name + "...");//sent the message to the client

//search by the given name, can be a machine name or text representation

InetAddress machine = InetAddress.getByName(name);//make an InetAdress object

//based on the IP gives the host name

out.println("Host name : " + machine.getHostName()); // sent the hostname to client with the exact message

//used the method toText

out.println("Host IP : " + toText(machine.getAddress())); // sent the host IP to client with the exact message

} catch (UnknownHostException ex) {

out.println("Failed in atempt to look up " + name);//try /catch in case the name can not be recognized

}

}

//same method with client

static String toText(byte ip[]) { //take as parameter the ip in byte , from -128 and 127

StringBuffer result = new StringBuffer();//contains a particular sequence of characters

for (int i = 0; i < ip.length; ++i) {//the for is until the length of the ip given as a parameter

if (i > 0) //if positive

result.append(".");

result.append(0xff & ip[i]); //0xff positive int , leave the 8 lower bits, add the ip[i]

}

return result.toString(); //return the updated string buffer object

}

}

/\*\*

\* The server is waiting for incoming connections from clients and uses the class ava.net.ServerSocket

\* \*/

public class InetServer {

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

int requests = 8; //number of client requests

int port = 5555;//the port that the server is waiting for connection with client

Socket sock;

ServerSocket server = new ServerSocket(port, requests);//the socket take as parameter the port and request number and we create an object

System.out.println("Athinas Inet server 1.8 starting up, listening at port 5555.\n");//print message

while (true) {

//the object created from the server socket uses the accept method to block the program running until there is an incoming request

sock = server.accept(); //can accept 6 clients, returns an object reference to exchange messages with the clients

new Worker(sock).start(); //start thread and Worker class

}

}

}

………………………………………………………………………………………………………………………………………………………………….

/\*\*

\*Athina Verroiopoulou

\*1/9/2017

\*CSC435, InetClient

\*/

import java.io.\*;

import java.net.Socket;

/\*\*

\*The client class is made in order to create a connection with the Server class.

\*In order to achieve that we need a port and an IP address that will wait for the connection

\*The port and IP values are given as input for the creation of Java.net.Socket class

\*/

public class InetClient {

public static void main(String args[]) {

String serverName; //we give an IP address or a hostName for our server in a String form

if (args.length < 1) //in case the length of the given server name is less than one

serverName = "localhost"; //the default choice is the local host

else

serverName = args[0];// server name is equal with the String value that is given, for my machine "192.168.0.34"

System.out.println("Athina's Inet Client, 1.8.\n");//print the message

System.out.println("Using server: " + serverName + ", Port: 5555");//print the Server name(hostname or IP address) and the using port

/\*\*

\* BufferedReader is used to read text from a character input stream, it buffers characters.

\* In Java Sockets every program can read and write in a Socket with the use of Input and OutputStream

\* that is a similar procedure with files. It represents an input stream of bytes and is used for the client / server connection.

\* We use InputStreamReader(System.in) to pass simple messages to server. For objects we should use ObjectInputStream / ObjectOutputStream

\*\*/

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));//read from Socket

//BufferedWriter out = new BufferedWriter(new OutputStreamWriter(System.out)); ( write )

try {

String name=null;//a string that is used to save host name or IP that is read

do {

System.out.print("Enter a hostname or an IP address, (quit) to end: ");//print the message

System.out.flush();//is used to empty the memory from every byte that is written in the buffer

name = in.readLine();//reads the host name or IP , in client, is given "192.168.0.34"

if (name.indexOf("quit") < 0)//if the response doen't return a word with "quit"

getRemoteAddress(name, serverName); //call the getRemoteAddress method that take for parameter the port and IP address

} while (name.indexOf("quit") < 0); // this will run until the user give an exit message , a word with "quit " inside

System.out.println("Cancelled by user request.");//print exit message

} catch (IOException x) {

x.printStackTrace();

}

}

//is used to produce a portable format for 128 bit

static String toText(byte ip[]) { //take as parameter the ip in byte , from -128 and 127

StringBuffer result = new StringBuffer();//contains a particular sequence of characters

for (int i = 0; i < ip.length; ++i) {//the for is until the length of the ip given as a parameter

if (i > 0)//if positive

//in the ip "192.168.0.34" add the "."

result.append(".");

result.append(0xff & ip[i]); //0xff positive int , leave the 8 lower bits, add the ip[i]

}

return result.toString();//return the updated string buffer object

}

//this method is called from the main to get the remote address from server

static void getRemoteAddress(String name, String serverName) {

Socket sock;

BufferedReader fromServer;

PrintStream toServer;

String textFromServer; //a string to save server answer

try {

sock = new Socket(serverName, 5555); //connection with server

fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));//read from the server, connect with socket

toServer = new PrintStream(sock.getOutputStream());//write / send to server, connect with socket

//The server is waiting for client response, the client will send the name of the host name or IP

toServer.println(name); //in this method the client will send to server the name that is passed as a parameter in getRemoteAddress(String name)

toServer.flush();//clean the memory

//if its not empty we will have 3 responses from server, one by one

//they should be:

//Looking up: name ...

//Host name : hostname

//Host IP : host ip

for (int i = 1; i <= 3; i++) {//will read 3 times

textFromServer = fromServer.readLine();//the client reads the server message and save it in textFromServer

if (textFromServer != null)//if the server answer is not empty

System.out.println(textFromServer);//print server message

}

sock.close(); //we call the close method of Java.net.Socket class to terminate the connection

} catch (IOException x) {

System.out.println("Socket error.");//the try catch is used to identify connection errors

x.printStackTrace();

}

}

}