/\* 2012-05-20 Version 2.0  
  
NAME: ROHAN DHOYDA  
  
Thanks John Reagan for updates to original code by Clark Elliott.  
  
Modified further on 2020-05-19  
  
-----------------------------------------------------------------------  
  
Play with this code. Add your own comments to it before you turn it in.  
  
-----------------------------------------------------------------------  
  
NOTE: This is NOT a suggested implementation for your agent platform,  
but rather a running example of something that might serve some of  
your needs, or provide a way to start thinking about what YOU would like to do.  
You may freely use this code as long as you improve it and write your own comments.  
  
-----------------------------------------------------------------------  
  
TO EXECUTE:   
  
1. Start the HostServer in some shell. >> java HostServer  
  
1. start a web browser and point it to http://localhost:4242. Enter some text and press  
the submit button to simulate a state-maintained conversation.  
  
2. start a second web browser, also pointed to http://localhost:4242 and do the same. Note  
that the two agents do not interfere with one another.  
  
3. To suggest to an agent that it migrate, enter the string "migrate"  
in the text box and submit. The agent will migrate to a new port, but keep its old state.  
  
During migration, stop at each step and view the source of the web page to see how the  
server informs the client where it will be going in this stateless environment.  
  
-----------------------------------------------------------------------------------  
  
COMMENTS:  
  
This is a simple framework for hosting agents that can migrate from  
one server and port, to another server and port. For the example, the  
server is always localhost, but the code would work the same on  
different, and multiple, hosts.  
  
State is implemented simply as an integer that is incremented. This represents the state  
of some arbitrary conversation.  
  
The example uses a standard, default, HostListener port of 4242.  
  
-----------------------------------------------------------------------------------  
  
DESIGN OVERVIEW  
  
Here is the high-level design, more or less:  
  
HOST SERVER  
 Runs on some machine  
 Port counter is just a global integer incrememented after each assignment  
 Loop:  
 Accept connection with a request for hosting  
 Spawn an Agent Looper/Listener with the new, unique, port  
  
AGENT LOOPER/LISTENER  
 Make an initial state, or accept an existing state if this is a migration  
 Get an available port from this host server  
 Set the port number back to the client which now knows IP address and port of its  
 new home.  
 Loop:  
 Accept connections from web client(s)  
 Spawn an agent worker, and pass it the state and the parent socket blocked in this loop  
   
AGENT WORKER  
 If normal interaction, just update the state, and pretend to play the animal game  
 (Migration should be decided autonomously by the agent, but we instigate it here with client)  
 If Migration:  
 Select a new host  
 Send server a request for hosting, along with its state  
 Get back a new port where it is now already living in its next incarnation  
 Send HTML FORM to web client pointing to the new host/port.  
 Wake up and kill the Parent AgentLooper/Listener by closing the socket  
 Die  
  
WEB CLIENT  
 Just a standard web browser pointing to http://localhost:4242 to start.  
  
 -------------------------------------------------------------------------------\*/  
  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintStream;  
import java.net.ServerSocket;  
import java.net.Socket;  
*/\*\*  
 \* HostServer Notes: This went pretty smoothly for me, although I did have to edit the HTML functions  
 \* to get an accurate content length so things would be compatible with browsers other than IE. I also modified  
 \* things to eliminate inaccurate state numbers based on fav.ico requests. If the string person wasnt found,  
 \* the requests was ignored  
 \*/  
  
/\*\*  
 \* AgentWorker  
 \*   
 \* AgentWorker objects are created by AgentListeners and process requests made at the various  
 \* active ports occupied by agentlistener objects. They take a request and look for the string  
 \* migrate in that request(supplied from a get parameter via an html form). If migrate is found,   
 \* the worker finds the next availabel port and switches teh client to it.   
 \*   
 \* I made a small modification because my browser kept requesting fav.ico. So I verified that we receive  
 \* a person attribute before processing the request as valid(and incrementing agent state)  
 \*  
 \*/*class AgentWorker extends Thread {  
   
 Socket sock; //We are here creating a server to initiate connection with client  
 agentHolder parentAgentHolder; // This will help maintaing agentstate as well as will hold the socket and state counter  
 int localPort; // It is declaring an interger variable local port and is likely to assign the port number that the server will listen for incoming network connections.  
   
 // Here we are creating a basic constructor. The purpose of this constructor is to  
 // initialize an instance of the AgentWorker class with the specified socket, port number, and agentHolder object.  
 AgentWorker (Socket s, int prt, agentHolder ah) {  
 sock = s; // The socket ensure network connection between server and client  
 localPort = prt; // The int parameter prt is used to define local port  
 parentAgentHolder = ah; //The ah represents clients or agent  
 }  
 // The reason we will be using run method the perform network communication between client and agent  
 public void run() {  
   
 //initialize variables  
 PrintStream out = null; //The out is a printstream object which will send data to the client  
 BufferedReader in = null; // The in a buffered reader object will read data from the client  
 //For this assignment the server code is hardcoded and this is just for basic implementation  
 String NewHost = "localhost"; //The new host will specify name of the host which is local host  
 int NewHostMainPort = 4242; // The local port will mainly run on port 4242  
 String buf = ""; // The buf method will help store in\coming data from the client  
 int newPort; // This variable will now store the new port number for the client  
 Socket clientSock; // This represents client socket  
 BufferedReader fromHostServer; // This will help host server to sead data from the client  
 PrintStream toHostServer; // It will send data to the host server  
   
 try {  
 out = new PrintStream(sock.getOutputStream()); // It is implemented for writing output stream of the socket  
 in = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
 // It reads from inputstream of the socket  
  
 String inLine = in.readLine(); // This a will read a line from the client  
 //Precisely determined the content to enable for usage on non-IE browsers.  
 //length and as a result need to build the html response so i can determine its length.  
 StringBuilder htmlString = new StringBuilder(); // It will build an html response  
 System.out.println();  
 System.out.println("Request line: " + inLine); // Print out a request line received from the client  
   
 if(inLine.indexOf("migrate") > -1) {  
  
 //If the client enters the string "migrate", it will create a new socket connection to the new server  
  
 //We will now create a new socket with the main server waiting on 4242  
 clientSock = new Socket(NewHost, NewHostMainPort);  
 fromHostServer = new BufferedReader(new InputStreamReader(clientSock.getInputStream()));//Read Input Stream of New Socket connection  
 //The code will send a request message to new server port on 4242 and it receive next open port for the client to connect to  
 toHostServer = new PrintStream(clientSock.getOutputStream()); //Will write the output stream of the 'client sock' socket connection  
 toHostServer.println("Please host me. Send my port! [State=" + parentAgentHolder.agentState + "]"); //Concatenating two strings together  
 toHostServer.flush(); //The data sent to the new server is actually transmitted through socket connection  
   
 //wait for the response and read a response until we find what should be a port  
 for(;;) {  
 //read the line and check it for what looks to be a valid port  
 buf = fromHostServer.readLine(); //Read a line of text from the input stream of client sock socket connection  
 if(buf.indexOf("[Port=") > -1) {  
 //If the variable contains the string, then the loop will be broken making use of break statement  
 break;  
 }  
 }  
  
 String tempbuf = buf.substring( buf.indexOf("[Port=")+6, buf.indexOf("]", buf.indexOf("[Port=")) ); //It extracts a substring from a larger text string  
 newPort = Integer.parseInt(tempbuf); // The value of the string is stored in tempbuf to an integer value and it will assign it to the new port variable  
 System.out.println("newPort is: " + newPort); // This will print new port  
 //HTML response is now prepared that will send to the user  
 htmlString.append(AgentListener.sendHTMLheader(newPort, NewHost, inLine));  
 //The user will now be acknowledged that the migration request was received  
 htmlString.append("<h3>We are migrating to host " + newPort + "</h3> \n");  
 htmlString.append("<h3>View the source of this page to see how the client is informed of the new location.</h3> \n");  
 htmlString.append(AgentListener.sendHTMLsubmit()); // This will return a string which contains html code  
   
  
 System.out.println("Killing parent listening loop.");  
 ServerSocket ss = parentAgentHolder.sock; // This will hold server socket object, that is used to listen for incoming network connections  
 ss.close(); //We will now close the port  
   
   
 } else if(inLine.indexOf("person") > -1) {  
 //increment the state int to reflect an event occuring in the 'game'  
 parentAgentHolder.agentState++; //If a string person will be found in the user's input then agentState will be incremented by 1  
 htmlString.append(AgentListener.sendHTMLheader(localPort, NewHost, inLine)); //the html will be send back to the user displaying agent state and form  
 htmlString.append("<h3>We are having a conversation with state " + parentAgentHolder.agentState + "</h3>\n");  
 htmlString.append(AgentListener.sendHTMLsubmit());  
   
 } else {  
 //When it is not able to find string person, this code will assume that the user will be requesting for a favicon  
 //The code will now generate an html response stating that their request is invalid  
 htmlString.append(AgentListener.sendHTMLheader(localPort, NewHost, inLine));  
 htmlString.append("You have not entered a valid request!\n");// The code will print " You have not entered a valid request"  
 htmlString.append(AgentListener.sendHTMLsubmit());   
   
   
 }  
  
 AgentListener.sendHTMLtoStream(htmlString.toString(), out);  
 sock.close(); //We will now close the socket  
   
   
 } catch (IOException ioe) {  
 System.out.println(ioe);  
 }  
 }  
   
}  
  
  
class agentHolder {  
 ServerSocket sock; //Listen for incoming client connection and will create new socket object  
 int agentState; //Integer variable for agentstate  
   
 //basic constructor  
 agentHolder(ServerSocket s) { sock = s;} //This is a constructor for 'agentHolder' class that will take Server Socket object as parameter  
}  
*/\*\*  
 \* AgentListener objects watch individual ports and respond to requests  
 \* made upon them(in this scenario from a standard web browser); Craeted   
 \* by the hostserver when a new request is made to 4242  
 \*  
 \*/*class AgentListener extends Thread {  
 //instance vars  
 Socket sock; //Communicate with the client that has connected to the server  
 int localPort; //The varaiable will store local port number  
   
 //basic constructor for AgentListener  
 AgentListener(Socket As, int prt) {  
 sock = As;  
 localPort = prt;  
 }  
 int agentState = 0; //An integer variable agentstate set to the default value of 0  
   
 //called from start() when a request is made on the listening port  
 public void run() {  
 BufferedReader in = null; //The in has been set to initial value of null  
 PrintStream out = null; // The out has been set to initial value of null  
 String NewHost = "localhost"; // The string variable new host is local host  
 System.out.println("In AgentListener Thread");  
 try {  
 String buf;  
 out = new PrintStream(sock.getOutputStream()); //A new printstream object is created and than it will send output to the client  
 in = new BufferedReader(new InputStreamReader(sock.getInputStream())); // the buffered reader object in is used to read input from the client  
 buf = in.readLine(); //This will read line of input from the client  
   
  
 if(buf != null && buf.indexOf("[State=") > -1) {  
 String tempbuf = buf.substring(buf.indexOf("[State=")+7, buf.indexOf("]", buf.indexOf("[State=")));  
 agentState = Integer.parseInt(tempbuf); //The ParseInt is actually known as an integer class and it will take string as an Argument  
 System.out.println("agentState is: " + agentState);  
   
 }  
   
 System.out.println(buf); // It will print the value of buf  
 StringBuilder htmlResponse = new StringBuilder(); //It will make an html response which will be sent back to client  
 htmlResponse.append(sendHTMLheader(localPort, NewHost, buf));//An HTML header is added to the htmlResponse object by this line of code. The method is called with three arguments local port, NewHost and buf  
 htmlResponse.append("Now in Agent Looper starting Agent Listening Loop\n<br />\n"); //The string will contain the following text as a html response  
 htmlResponse.append("[Port="+localPort+"]<br/>\n");  
 htmlResponse.append(sendHTMLsubmit());  
 //display it  
 sendHTMLtoStream(htmlResponse.toString(), out); //This method takes two arguments. A string which is representing an html response and an qoutput stream  
   
  
 ServerSocket servsock = new ServerSocket(localPort,2);// This code will help opening connection at a port  
 agentHolder agenthold = new agentHolder(servsock);// A new agentState is created which will store the socket and agent state  
 agenthold.agentState = agentState;  
   
 //wait for connections.  
 while(true) {  
 sock = servsock.accept(); // A server socket has been created  
 System.out.println("Got a connection to agent at port " + localPort);  
 //connection received. create new agentworker object and start it up!  
 new AgentWorker(sock, localPort, agenthold).start(); //A new Agent worker has been created  
 }  
   
 } catch(IOException ioe) {  
 //this happens when an error occurs OR when we switch port  
 System.out.println("Either connection failed, or just killed listener loop for agent at port " + localPort);  
 System.out.println(ioe);  
 }  
 }  
 //send the html header but NOT the response header  
 //otherwise same as original implementation. Load html, load form,  
 //add port to action attribute so the next request goes back to the port  
 //or goes to the new one we are listening on  
 static String sendHTMLheader(int localPort, String NewHost, String inLine) {  
 // The SendHTMLheader is taking three arguments localPort, NewHost and inLine  
 StringBuilder htmlString = new StringBuilder(); // This will initialize a stringbuilder object which will construct html string in memory  
 htmlString.append("<html><head> </head><body>\n"); //This will start a new HTML document and will contain the opening of <html>,<head> and <body> tags  
 htmlString.append("<h2>This is for submission to PORT " + localPort + " on " + NewHost + "</h2>\n"); // The HTML port is submitted to localport on a specific port i.e NewHost  
 htmlString.append("<h3>You sent: "+ inLine + "</h3>");//This code concatenate the value of the inLine variable  
 htmlString.append("\n<form method=\"GET\" action=\"http://" + NewHost +":" + localPort + "\">\n");  
 htmlString.append("Enter text or <i>migrate</i>:");  
 htmlString.append("\n<input type=\"text\" name=\"person\" size=\"20\" value=\"YourTextInput\" /> <p>\n");  
   
 return htmlString.toString();  
 }  
 //finish off the html started by sendHTMLheader  
 static String sendHTMLsubmit() {  
 return "<input type=\"submit\" value=\"Submit\"" + "</p>\n</form></body></html>\n";  
 }  
 //The content length is calculated and the response header is send so that we can play a nicer game with all the browsers  
 //and it can actually work with any non ie browser  
 static void sendHTMLtoStream(String html, PrintStream out) {  
 // A static method which will send a html string to a print Stream object  
 out.println("HTTP/1.1 200 OK"); //It will indicate that the server is responding  
 out.println("Content-Length: " + html.length()); //It will determine the length of the html string that will be sent  
 out.println("Content-Type: text/html"); //It is a required header file for HTTP responses of how the data in response should be interpreted  
 out.println("");   
 out.println(html);  
 }  
   
}  
  
public class HostServer {  
 public static int NextPort = 3000; //The code will start listening at port 3001  
   
 public static void main(String[] a) throws IOException {  
 int q\_len = 6; //Maximum number of simultaneous client connection on a server  
 int port = 4242;//The Socket Server listens for incoming connections on a specified port 4242  
 Socket sock;  
   
 ServerSocket servsock = new ServerSocket(port, q\_len); //This method assigns a servsock variable, using specified port number and queue length  
 System.out.println("Elliott/Reagan DIA Master receiver started at port 4242.");  
 System.out.println("Connect from 1 to 3 browsers using \"http:\\\\localhost:4242\"\n"); //The user will connect to a local Server using three different web browsers  
 while(true) {  
 NextPort = NextPort + 1; //We will increment to the nextport  
  
 sock = servsock.accept(); //Accepts an incoming and it will create a new socket object  
 System.out.println("Starting AgentListener at port " + NextPort); //create new agent listener at this port to handle incoming request  
 new AgentListener(sock, NextPort).start(); //The AgentListener object will be created at the new port number to handle incoming request  
 }  
   
 }  
}