ID1212

The Hangman Game

This is the report for homework one in the course ID1212 Network programming

Rafi Malkhasian
6-4-2019

Table of Contents

Introduction

Network programming is one of the important sections in allowing communication between two different devices. In this report we will use the server client model to write an application connecting a user to a server. Furthermore, the server will allow the user to play the Hangman game and saving the score and the result of each of the clients.

This communication app will use blocking sockets to send and receive data to and from the client. A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is going to be sent to.

The hangman game is a classical game where a person has to guess a word with a given length. The user also has the choice to guess the entire word or a letter that may be in the word. The user is allowed only a certain number of tries and if the user did not guess the word within these number of tries, the user loses a point.

To allow multiple users to connect to the same server we use multithreading and parallel programming techniques that enables the server to handle multiple requests simultaneously. In addition, the application will be divided into the MVC structure for better maintenance and future development. MVC will also make the code easier to read and providing good encapsulation.

Another requirement in this assignment is that server should not send any part of the view, such as a string or an integer. Instead, the server should send state that is inserted in the client's interface. Furthermore, the client should not store and of these states as the client should only have an interface that prints the state.

An important part of the implementation requirement is that the user should be able to send commands to the server without the need to wait for the response from the server. This means the client side should be multithreaded allowing the client to receive and send messages to the server simultaneously. A similar approach is required also for the server side as the server should be able to handle multiple clients at the same time.

Literature Study

The lecture videos were used in this assignment along with the books "An Introduction to Network Programming with Java, 3rd Edition, by Jan Graba, Springer, 2013, ISBN: 978-1-4471-5253-8 (Print) 978-1-4471-5254-5" and "Java Network Programming, 4th Edition, Developing Networked Applications, by Elliotte Rusty Harold, O'Reilly & Ass., Inc., 2013"

Method

One of the requirements in this seminar is to have a good architecture where it is easy to understand the code and maintain. An easy way to achieve this is by following the MVC structure. Therefore, we start by creating the server folder and the client folder. These two folders will contain the files related to each side of the communicating entities. The client side will only provide the view for the user to view the messages from the server. In addition, the client is should be able to send a message to the server. Therefore, it is more convenient to have the users input and output in a subfolder for easier management of the application. On the other hand, the server side will be further divided to the subfolders controller, model, startUp. The startUp folder is responsible for starting the application and creating the server. The server will also have an own controller that's placed in the controller folder. The server model will create the socket which the client connects to. Once the socket is created, the server is put into a blocking state waiting for a client. Once a client is connected, the server controller will create a thread to handle that specific client. Therefore, we need another controller responsible only for handling requests from one client. In other words, a thread is created on the server side each time a client is connected to the server to handle the requests of that client. Once the connection is closed, the thread stops but the server is still running. The hangman game model is placed also on the server side because we don't want the client side to be concerned about the implementation details thus making an easy interface to use. In turn, this will provide an interface for the client printing the reply from the server without storing and variables or states.

As mentioned before, the connection between the client and the server is done using TCP blocking sockets. To achieve this goal, both the client and the server connect over the same port and the appropriate type of socket. The server side use's a socket od type ServerSocket which is imported from the package java.net and the client uses the type Socket. The client's Socket in this program takes the host address (which is the local host at the moment) and the port number that matches the same port number on the server. Using a ServerSocket type allows us to use the method accept() to take incoming connection requests from clients. With this we have established a TCP connection between the server and the client.

The client side, as mentioned before, is provided with only and interface. In this program we can see that the client has only two threads for reading and writing. The first thread is the receiver which is responsible for writing the replies from the server to the client's screen. The second thread is the sender which takes the input from the client and sends it to server. As the requirements specify, the client side does not store the reply from the server nor any state but instead only prints and sends messages.

Another feature in this program is that the server sends only state to the client and not a particular part of the view. This is an indication of a good design and good encapsulation as it reduces the risk of the output being manipulated or altered. In addition, having these states stored on the server allows easier track of the user's input and providing a safer responsive interface for the client. In this program, the server sends a serialized object which contains the result of the user's guess such as the attempts left and the client's score. This object is deserialized and printed on the client side.

As mentioned earlier, we have seen how the client side is implemented using threads which allows the client to send multiple messages and not having to wait for a reply from the server. This is very helpful in

situations where the server takes a long time to send a response and the client can send a request to close the connection. The client in this program could just write ***CLOSE*** to close the connection.

Results

We test the results by running the server side first and then start the client side. When the server starts running, it is put in a blocking mode waiting for a client to connect. Each time a client is connected in the specified port, the controller creates a thread to handle the requests of that client. In turn, the server can handle many clients at once and does not wait for the one client to finish to connect to the next. When a person establishes a connection to the server, the client can start the game and play as many times as desired. The score is updated after each game. The client can also quit any time just by typing "***CLOSE****". In doing so, the thread handling the current client dies after the connection is closed while the server is still running waiting for new clients.

When the client sends a message to the sever, the server creates a serialized object which is sent to the client's interface to be printed. And as we can see the client does not store any of the states that is sent by the server. An important notice here is that the Guess object which is sent to the client also contains another serialized object which stores the correct letters that the client guessed. This object is the CorrectLetter and the when the guess is incorrect, the attempts left for the client is decreased.

Discussion

This program shows a good structure following the MVC pattern and having high cohesion and low coupling. This program also shows potential of further development such as implementing more games on the server allowing clients to choose the desired game. A very important note about the architecture in this program is that in the main method of the server, a server model is created and the server in turn, creates the controller. The reason for this implementation is to provide good encapsulation and high cohesion. To explain further, this implementation provides the server model with its own controller that could be further developed to handle connection problems and close unsecure connections. This means, we can view the server side to be composed of several entities and the server model being one of those entities having its private controller.

Another suggestion is to have a master controller for the entire server side that could be used to start the server model and initiate database connection. However, the program currently implemented is not that big and doing all those modifications will take extra space that we could avoid. All this could provide an easier understanding of the code and keeps a low coupling.

Finally, we can see that with future development we can implement a view for the server side too and providing an interactive interface for the administrator. In addition, we can see that the program shows a great potential for future development with minor modifications. The user interface could also be improved to include more games and friendlier interaction with the client.