|  |  |  |  |
| --- | --- | --- | --- |
| Version: 1 | Before you start to work with this template read the instructions (see link) “How to Use this Template” on the first page | | |
| Author: Sung, Minjung | **Software Design Description (SDD)** | | |
|  | Open chatting programming | | |
|  | | |
|  | **History** | | |
|  | Version  Enter document version; (see rules below) | Date  Enter document version date 01-JUN-2005 | Status (Comments)  Draft, For Review, Released.  Comments are needed if a SDD has several “releases” |
|  | 0.1 | 01.01.2021 | Initial draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

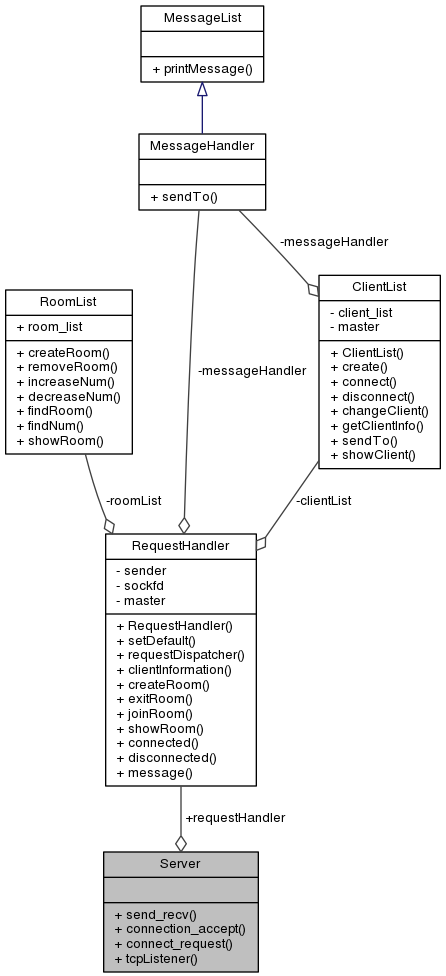
# Overview and Design

## Overview/Introduction

This chat programming is web-based open chat programming in order to provide online open chat environment to users without much credential required. The program provides four different simple commands: “createRoom”, “exitRoom”, “joinRoom”, and “showRoom”. These commands need to be typed exactly (case sensitive). Each command is implemented in RequestHandler class for its own use.

## Design

### Server



#### Function Explanation

* Server

|  |  |
| --- | --- |
| Function Name | Description |
| tcpListener() | **input**: int sockfd, int fdmax, struct sockaddr\_in my\_addr, struct sockaddr\_in client\_addr, fd\_set master, fd\_set read\_fds  **output**: void  tcpListener repeatedly checks requests (either connection or others) coming from clients using while(1) |
| connect\_request() | **input**: int\* sockfd, struct sockaddr\_in\* my\_addr  **output**: void  connect\_request executes   1. setsockopt() 🡪 it allows an application to reuse the local address when the server is restarted before the required wait time expires 2. bind() 🡪 it supplies a unique name for the socket 3. listen() 🡪 it allows the server to accept incoming client connections |
| connection\_accept() | **input**: int i, fd\_set\* master, int sockfd, int fdmax  **output**: void  connection\_accept checks whether the connection is correctly made. If so, requestHandler receives “connected” message with sockfd. If not, print an error. |
| send\_recv() | **input**: int i, fd\_set\* master, int sockfd, int fdmax  **output**: void  If recv function receives anything without any errors, it returns a size of char pointer.  recv\_buf > 1: passes the received char pointer to requestHandler as string  recv\_buf <= 0: print an error |

- RequestHandler

|  |  |
| --- | --- |
| Function Name | Description |
| setDefault() | **input**: int sockfd, fd\_set\* master  **output**: void  set global variables using sockfd and master member variables. These information needs to be passed onto messageHandler for socket communication |
| requestDispatcher() | **input**: int sender, string request  **output**: void  requestDispatcher calls other functions in RequestHandler class based on request in a form of string. Those cases are   1. clientInformation 2. createRoom 3. exitRoom 4. joinRoom 5. showRoom 6. connected 7. disconnected 8. and others 🡪 assumes these are just regular chat messages |
| clientInformation() | **input**: int sender, string request  **output**: void  cut “clientInformation:” part in front of the actual client information and send it to clientList.create() as member variable |
| createRoom() | **input**: int sender  **output**: void  createRoom() command is only available to clients in lounge (this information can be confirmed with getClientInfo(sender,ROOMNUMBER)).  true (in lounge) 🡪 search for any skipped room in roomList. If there is a skipped roomNumber, create a room for it. If not, create a room consecutively after the lastly created one.  false 🡪 print an error message saying that it’s not possible to create new room |
| exitRoom() | **input**: int sender  **output**: void  This removes a request sender from a chatroom to lounge. Get clientInfo regarding his/her roomNumber and check whether the sender is in lounge or not.  true 🡪 exitRoom command invalid since the sender is already in lounge (error message returned)  false 🡪 decrease num for the joined room, increase num for lounge, and change his/her roomNumber in clientInfo |
| joinRoom() | **input**: int sender  **output**: void  Let the client join already existing room in roomList. Inquire him/her which room to join and loop through roomList to see whether map<int,int>roomList has the requested room in its keys  true 🡪 increase num for the requested room, decrease num for lounge, change clientInfo for roomNumber(index 2)  false 🡪 return an error message saying that the requested room doesn’t exist |
| showRoom() | **input**: int sender  **output**: void  Passes an int index 10, which simply refers to print a message (roomList.showRoom()), onto messageHandler.sendTo() |
| connected() | **input**: int sender  **output**: void  If the connection is correctly made from server, server calls this function. This checks the existence of lounge and increase num for it. If not, create one |
| disconnected() | **input**: int sender  **output**: void  If a client is disconnected for any reasons, decrease number for the original room.  Case: where other people still in the original room and the original room is not lounge 🡪 send leave message to them  Call clientList.disconnect() function to change socket and room number |
| message() | **input**: int sender, string request  **output**: void  In a case where the request doesn’t fit string comparisons listed above, assumes it to be regular messages, so pass the request to messageHandler.sendTo() with username of the sender in front. |

* ClientList

|  |  |
| --- | --- |
| Function Name | Description |
| create() | **input:** int sender, string ci  **output:** bool  string ci will come in as  ex) minjung, password123  Therefore, initialize vector<string> information and push\_back this information by separating it with comma. Once it’s done, push\_back room (always 0 == LOUNGE) and sockfd (same as sender) number additionally. |
| connect() | **input:** int sender, string ci  **output:** bool  connect() is a lot like create(), yet this one differs from create() in an aspect of that it’s for already existing client in clientList and create() is for creating a new account.  It checks whether the given username and password match to any of components in clientList using map<int, ClientInfo>::iterator.  true 🡪 create ClientInfo with new sockfd(== sender) and room (== LOUNGE) number, remove already existing information in client\_list, and insert new clientInfo instead.  Used erase/insert instead of setSockfd()/setRoomNumber() because I couldn’t come up with a way to replace a key value in a map<int, ClientInfo> |
| disconnect() | **input:** int sender  **output:** bool  Disconnect a client by changing sockfd and roomNumber to “-1”. Since this also needs to be applied in a map and I don’t know how to replace key value in a map, I used client\_list.erase()/client\_list.insert() instead, again. |
| changeClient() | **input:** int sender, int index, string change  **output:** bool  changeClient handles 4 different cases (username, password, roomNumber, and sockfd). As member variables, it receives index (which one to change) and change (changes to what) when the sender is found in client\_list.  One thing to notice is that sockfd change should also be applied to map<int,ClientInfo> because the key value represents sockfd.  If the index doesn’t match any of cases, messageHandler prints an error message |
| getClientInfo() | **input:** int sender, string ci  **output:** string  Find sender in client\_list and return its corresponding information as string  \*\*index\*\*  0 🡪 username  1 🡪 password  2 🡪 roomNumber  3 🡪 sockfd |
| sendTo() | **input:** int sender  **output:** bool  sendTo() loops through client\_list, extracts socket number that has the same roomNumber as sender’s, push\_back them into vector<int> sendList, and returns the vector as result |
| showClient() | **input:** void  **output:** string  showClient() returns socket number and username for all clients in client\_list as string |

* RoomList

|  |  |
| --- | --- |
| Function Name | Description |
| createRoom() | **input:** void  **output:** int  int room increments one by one and checks whether room\_list already contains that roomNumber in the list. Once it finds a roomNumber that’s not included in room\_list, create a room for the corresponding number using emplace(roomNumber, 1) |
| removeRoom() | **input:** int roomNum  **output:** void  removeRoom() simply removes for the roomNum in map<int,int> |
| increaseRoom() | **input:** int roomNum  **output:** void  increaseRoom() increases 1 value for the matching roomNum in map<int,int>. find(roomNum) will return a pointer to the target roomNum, 🡪second will give the number of people in that chatroom, and increaseRoom() adds one to that value |
| decreaseRoom() | **input:** int roomNum  **output:** void  decreaseRoom() decrease 1 value for the matching roomNum in map<int,int>. The logic for how to do so is same as increaseRoom() |
| findRoom() | **input:** int roomNum  **output:** bool  findRoom(int roomNum) simply checks whether roomNum exists in room\_list and returns a result of the check |
| findNum() | **input:** int roomNum  **output:** int  findNum() provides the number of people in roomNum.  true 🡪 the number of people in roomNum  false 🡪 -1, which is impossible. In this case, just assumes to be an error |
| showRoom() | **input:** void  **output:** string  Loop through map<int,int> using iterator, add up all components in room**\_**list, and return the string as a result |

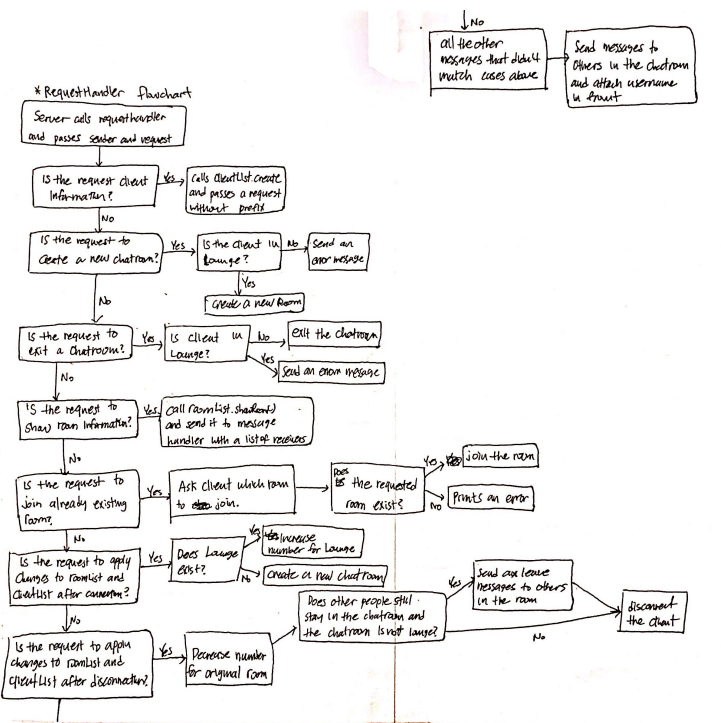
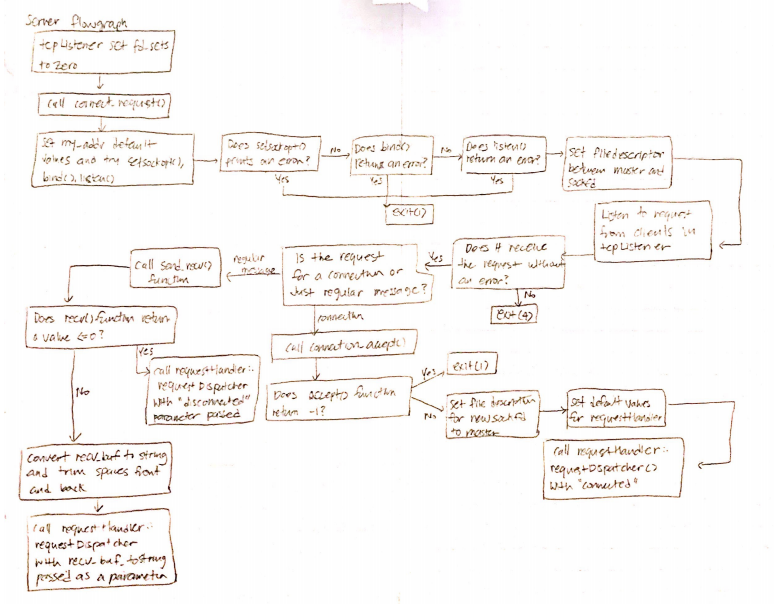
* MessageHandler

|  |  |
| --- | --- |
| Function Name | Description |
| sendTo() | **input**: fd\_set\* master, int sender, vector<int> receiver, int messageChoice, string param  **output**: void  sendTo() receives receiver as vector<int> and send messages to the ones in the list. There are two cases: vector<int> size == 1 and others.  case 1 (size == 1): send the message to the sender  case 2 (size => 1): send messages others except sender  messageChoice has its matching values listed in MessageList. One thing to notice is that when messageChoice is “10”, this assumes it to be a regular message and simply sends param as a whole message instead of choosing a message from messageList |

* MessageList

|  |  |
| --- | --- |
| Function Name | Description |
| printMessage() | **input**: int m\_choice, string m\_param  **output**: string  printMessage function receives m\_choice and m\_param as member variables and returns the equivalent message of the m\_choice, added with m\_param, as a string\_ |

#### Flow Chart



#### Sequence Diagram

**command**

title Sequence Diagram

participant Server

participant Client

participant RequestHandler

participant ClietList

participant RoomList

participant MessageHandler

Client->Client:[1] tcpListener()

Client->Client:[2] connect\_request()

Client->Server:[3] connect()

Server->Server:[4] tcpListener()

Server->Server:[5] connect\_request()

Server->Server:[6] socket()

box over Server:setsockopt()\n\nbind()\n\nlisten()

Client->Client:[7] send\_clientInformation()

Client->Client:[8] setClientInformation()

Client->Server:[9] send()

Server->Server:[10] connection\_accept()

Server->Server:[11] accept()

Server->RequestHandler:[12] setDefault()

Server->RequestHandler:[13] requestDispatcher("connected")

RequestHandler->RoomList: [14]: createRoom()

Server->Server: [15] send\_recv

Server->RequestHandler:[16] requestDispatcher("ClientInfo:")

RequestHandler->RequestHandler: [17] clientInformation()

RequestHandler->ClietList: [18] create()

RequestHandler->MessageHandler:[19] sendTo(sender)

