6.005 elements of software construction
Final project – IM message system

By Eric Lu

1. Definition of a conversation:

User can start a personal conversation by clicking the other desired user who is in online list on the GUI interface; meanwhile, a conversation will be created with a special ID after the message being sent from the user. Each conversation will have a list of current user. Users can also invite others to join the current chat room. For users who have multiple invitations, they can join as many chat-room/ conversations as he/she want. As long as no client is in the conversation, the conversation object will be destroyed.

2. Class implantation

Server-Side:

Server - handling the multi-user/client connection and pass it to client handler

- Fields:
    - HashMap < hash(username), UserInfo > OnlineUsers
    - HashMap <hash (Id), conversation > allConvs
    - ServerSocket : Socket
- Methods :
    - Main (port number): start a server with given hostname and port number.
    - Serve(): handling a new socket from client then throw it to a ClientHandler, new thread.
    - AddUser : add a new userifno to hashmap
    - RemoveUser : remove a userifno to hashmap
    - RemoveConv : add a new conversation after it is created
    - newConv: remove a conversation if no user is in it.

UserInfo – It contains user input/out handler and other user's information such as user id and conversations that a user participation.

- Field:
    - Outputhandler output
    - Inputhandler input
    - HashMap <Id, conversation > conversations

InputHandler extends Thread – a sub-server that handles the multiple client sockets. At first, it will check whether the username of client is available, if not, the client socket will be closed. It will parse socket's Input Stream to a Message object, and put a queue to BlockingQueue of conversation.

- Fields:
  - Socket: clientSocket
  - ObjectInputStream: in
  - HashMap<Interger , Conversation > MyConvs – conversations of the current connected user
- Methods:
  - ClientHandler(Socket) – constructor
  - handleRequest () – Performing action correspond to input Object.
  - checkName (String username) -- check whether the username is valid. If it is valid, the username will be added to user-list of Server.
  - CreateConversation ()– Create a conversation object if there is no such a conversation with given Id.
  - JoinConversation(Conversation) – Joining a specific conversation
  - LeaveConversation (Conversation) – Turning an input to leave the conversation Message.
  - updateMessage (Conversation) – Turning a input Stream String into a Message abstract data type to put onto shared queue with Conversation.
  - Run() : sits in a loop calling put() on the inputQueue

OutputHandler extends thread – Take output queue, turning Message ADT into a output stream to client such as other user leave the conversation or message updating from other user.

- Fields :
  - Socket: clientSocket
  - ObjectInputStream: out
  - BlockingQueue <Message> outputQueue -- produced by different conversation or the server thread
- Methods:
  - ClientHandler(Socket) – constructor
  - Run() : sits in a loop calling take() on the outputQueue , passing to handleRequest().
  - handleRequest (Message) – performing action correspond to output queue.
  - CreateConversation () –Notifying client that a new conversation has been established.

    - LeaveConversation (Conversation) – Notifying the client that the other user just left the conversation.
    - updateMessage (Conversation) – After receive message updating queue, parsing it to a string which can be write out to client.
    - updateQueue(Message) – put a message onto outputQueue

Conversation extends Thread – Model of the MVC, a mutable data-type that stores the messages between clients. Conversation will wait and take the queue, copying and putting correspond message queues onto client-output-handlers outputQueue.

- Fields:
    - BlockingQueue <Message> InputQueue
    - HashMap< username, OutPutHandler > Users of all clients in the conversation.
- Methods:
    - Conversation (BlockingQueue<message> message, Integer Id) : initialize a conversation with given id
    - Run() : sits in a loop calling take() on the inputQueue
    - HandleQueue(Message): Handing the message queue , turning into a action such as addClient or removeClient.
    - addClient (OutputHandler) : Conversation will access the clientOutputhandler from OnlineUser of server, adding a new client who subscribe to this conversation and notify all the other client in this conversation by putting a message to output handler's queue.
    - getCurrentUsers() : return the user name is the conversation.
    - RemoveCLient(OutputHandler) : remove client output handler in the conversation and notify all the other clients in the conversation.
    - updateMessage() : After consuming a queue from InputQueue, send the message back to each users client output handler in this conversation.
    - updateQueue(Message): add a queue to Conversation's output queue.

Client-Side:

User – At First, a user, controller part of MVC, will invoke a MainBoard-GUI. Client has to make a socket connection to the server with specific address, port number. Afterward the User(client) has to type username to server to check if it's valid. If the username is not valid or unique client are asked to type a new one. No conversation would start if the username is not valid.

- Fields :
    - Socket : clientSocket
    - ChatBoard : GUI
    - String : Username
    - ArrayList<String> OnlineUsers
    - ObjectInputStream : in
    - ObjectOutputStream : out
- Methods :
    - User(hostname, port, username): -- constructor
    - OpenChatBoard : Invoking the ChatBoard graphic user interface.

ChatBoard extends JFrame – View of the MVC , also a GUI that a user can see conversation with others and list of online users and communicate with other user. User can start a conversation with other users by triggering an event in the MainBoard ex. Clicking another user to start to chat.

- Fields:
    - JComponement : JTable, Jbutton, JLabel….
    - Conservation : conversation of the current user with others.
- Methods:
    - ChatBoard() : construct a GUI of char room

Messages transferring (ADT) between Client and Server :

UserInfo implements ToServerMessage – it is immutable and represents the user-Information object that client pass to server via socket. Containing the user information such as username, the id of a conversation that user join.

- Fields
    - String username
    - ArrayList< String > ChatRoomId
- Methods :
    - UserInfo(username, ChatRoomId) – constructor
    - addRoom () – add a new chatroom that user joined.
    - deleteRoom () – delete room that the user just leave.

ChatMessage implements ToServerMessage, ToClientMessage – representing a new message and user-info of the conversation with id specified from client.

- Fields:

- String ConversationId
- String updatingMessage
- String From -- username

- Methods :
  - ChatMessage(String username, String id, String text) – constructor
  - getUsername () : return the message owner.
  - getRoom () : return the id of conversation this message belongs to
  - getText (): return the content of the message.

UserOlineList implements ToClientMessage – it contains the list of current online user and their information from server. It will be sent to all online users while a new user is online or someone is offline

- Fields
  - ArrayList<String > OnlineUsers
- Methods :
  - getUserList() : return the online username

ToClietMessage interface – an interface that implemented by server to client and contains a method of converting a string to a ChatMessage object.

ToServerMessage interface – an interface that implemented by all clients to server message and contains a method that convert a string to a ChatMessage object

3. Protocol for communication between server and client

Client-to-server Message Protocol

```
PROTOCOL :: = (START | LEAVE | SENDMESSGAE| DISCONNECT| CONNECT)*
START :: = TO SPACE USERNAME+
LEAVE :: = EXIT SPACE CHATROOM
SENDMESSGAE :: = SEND SPACE MESSAGE SPACE CHATROOM
DISCONNECT ::= QUIT USERNAME
CONNECT ::= TELNET SPACE HOSTNAME PORT
HOSTNAME ::= [DIGIT (.?)]+[:] | TEXT+[:]
TELNET :: = " telnet "
PORT ::= DIGIT+
MESSAGE ::= (TEXT SPACE?)+
```

USERNAME ::=   ^[a-zA-Z0-9] (TEXT)*   SPACE

// A username should start with a-z or A-Z or digits and can also contain [-] or [_] or [.]

CHARTROOM ::= "id:" DIGIT

TO ::= "to"

EXIT ::= "exit"

SEND ::= "send"

QUIT ::= "quit"

Server-to-Off-Line-client Message Protocol:

Protocol :: = (WELCOME | DISCONNECT| UPDATEMESSAGE | JOINMESSAGE) *

WELCOME ::= "Hello !" SPACE USERNAME "you're connected to " HOSTNAME

DISCONNECT ::= "Sorry server " HOSTNAME " is currently unavailable. Please try again."

UPDATEMESSAGE ::= USERNAME   SPACE "say" SPACE (TEXT SPACE? NEWLINE?)+

JOINMESSAGE ::= USERNAME SPACE "join" SPACE CHATROOM

HOSTNAME ::= [DIGIT (.?)]+[:] | TEXT+[:]

CHARTROOM ::= "id:" DIGIT

USERNAME ::=   ^[a-zA-Z0-9] (TEXT)* SPACE

General :

TEXT ::= [^(NEWLINE|SPACE)]+

DIGIT :: = [0-9]

SPACE ::= " "   // an single character space

NEWLINE ::= "\n"

4. Concurrency strategy

On the server side there are 4 different kinds of threads one main thread that initialize the server and listen for incoming client, an input handler, an output handler, and a conversation thread. While clients make connection to the server, multiple input handers will process it at the same time. Input Handler is thread-safe because it passes a message queue to a blocking-queue of conversation. To prevent rare condition in the Server thread, read and write methods that change the OnlineUserList such as addUser , removeUser will be synchronized; besides, the operations that mutate the AllConvs will be synchronized as well. The Output handler will take a blocking queue put by Conversation or the server-main-thread; then it will write it out to the client. In the server thread.

In the conversation thread, it has 2 private fields one is thread safe blocking queue handling message of different client thread, the other one is a hash map containing output handler of the current user. Because the List of client output handler in the hash map can only be called and changed by the conversation thread itself, it can never interleave each conversation thread.

On the client side, aside from GUI being invoked by the main thread, ActionPerformers within GUI are thread-safe because it is handled by event dispatch thread, running asynchronously. The client will also have a background thread that receives message and modify the Model component of chatroom. As for sending output to the server, I will use action listener in the GUI thread to send message to the server when the appropriate action events occur.

5. Testing strategy

Server- side testing

- ■ Testing add a new user or conversation
    - ■ Make a multiple thread request that add or remove a Userinfo from the server's hashmap at the same time
    - ■ Make a multiple thread request at the same time that add or remove a conversation from the server's hashmap
    - ■ Make sure all output handler receive the proper queue after action mentioned above is performed
- ● Testing UserInfo
    - ◆ Make sure initiating input and output handler properly
    - ◆ Should be added into OnlineUserList if given username is unique
- ● Testing input handler
    - ■ Check that handling input does what is expected
        1. Try to create a conversation
        2. Check if username is valid and unique(with valid form name, not valid , unique , and not unique)
        3. Try to join a current existing/non existing conversation in the Server's AllConvs list.
        4. Try to leave a conversation (Both the user is in or not)
        5. Send a message to conversation (to the one user is in or not)
    - ■ Make sure putting queue onto a conversation properly
        - ◆ Creating a lot's of message queue for different conversation thread to make sure every conversation receive proper queue.
- ● Testing output handler

- ■ Check handling Request can parse the inputQueue properly
    1. Create a conversation (notify user who is in the current conversation and those who are not)
    2. Notify that a user has left the room (to those who are in the conversation or not)
    3. Notify that a user just sending some message to a conversation (to those who are in the conversation or not)
- ■ Take the blocking queue from server or conversation as expected
    - ◆ Create a lots of conversation thread and a server thread and put queue at once to make sure that not a single queue lost

- ● Testing Conversation object
    - ■ Make sure initiate a conversation with unique id properly
        - ◆ Make sure parse the message input queue properly
        1. Add a client output handler to UserList on conversation and make sure output handler in the UserList reveive it
        2. Remove all user in it, and check it is removed by the server
        - ◆ Make update queue works properly
        1. Create a lot's of client input handler thread to send message queue to conversation. Then make sure receive queue without losing anyone of it.

Client- side testing

- ■ Test individual chatting system
    1. Create a valid username and attempt to login
    2. Create an invalid username and check to see an indication if the username is taken or invalid
    3. Create a conversation with a valid conversation name
    4. Create a conversation with an invalid conversation name (bad characters/already taken)
    5. Join conversation
    6. Exit conversation
    7. Message all the users in the conversation
- ■ Test multiple tabbed chatting system
    1. Create and join multiple conversation and messages to different conversation
    2. Check for concurrency bugs for a particular client
- ■ Test multiple users in a conversation

1. Multiple users will message in the conversation
2. Users will join and exit the conversation and the output of the messages will be checked against expected conversation behavior

- Combine multiple chatrooms and multiple users
  1. Consider all cases of performing actions on the conversation, e.g. The lastperson exits the room and someone attempts to join the conversation at the same time.
- Deal with sudden disconnects
  1. Have a user join multiple conversations and then disconnect from the entire chat
- GUI testing
  1. Make sure gui components are updated when the client side model is updated
  2. Make sure that the display doesn’t freeze up no matter how many messages are being sent from the server (a lot!)
  3. Make sure that displays are dynamic; conversation list and user list updates without the need for refreshing
  4. Make sure all the components are sitting in the right place