**CS 3516 / CS 513 Class Project**

**A Client-Server Chat Program**

**Zishan Qin(654953308)**

**Date of Submission:07/02/2015**

This project is my personal work unless marked to the contrary.

Signature: Zishan Qin

# Abstract

This report outlines the design and development a chat program based on the client-server model. The program was written in java to run under the Mac OSX operating system on JVM. The design and program are modular in nature and make good use of abstract data types and of software re-use. The report includes test cases used to verify the correct operation of the program, as well as the entire code.

# Contents

Table of Contents

Abstract 2

Project Description 4

Detailed Design 5

Server-Side Design Architecture 5

Client-Side Design Architecture 7

Testing and Evaluation 9

Test Metrics 9

Server Test cases and results 9

Client Test cases and results 12

Stability and Error Reporting Test cases and results 14

Future Development 17

about GUI 17

About functions: 17

Conclusion 18

Appendix 19

Reference 19

Code 19

# Project Description

The project is to design and implement a chat program based on the client-server model using Java. MVC is used as the architectural pattern this project uses to organize the code and make the codes into three interconnected parts: model, controller and view, so as to separate internal representations of information from the ways that information is presented to or accepted from the client and server. Multi-thread programming, socket network programming skills and Swing GUI programming skills are used. The learning object of this project is provides me with insight into practical computer networks and the problems faced when implement them.

# Detailed Design

## Server-Side Design Architecture

Since there is no need to have a GUI for server, we just have Model and Controller part in our server side.

In model, we store the data of our models, which are varieties of types of messages processed by our handler.

|  |  |
| --- | --- |
| Class name | Responsibility |
| BroadCastMSG | Record message which will be broadcasted to all clients sent from one specific client. |
| ConnectMSG | Record message which represents the connection requested by a client |
| DisconnectMSG | Record message which describe the disconnection request by a client |
| EndToEndMSG | Record whisper message between one client to another |
| NameListMSG | Represent the list of all online users |
| UnexpectedDisconnectionMSG | Record the message which describe a disconnection due to varieties of Exceptions, such as IOExpection, TimeoutExpection etc. |

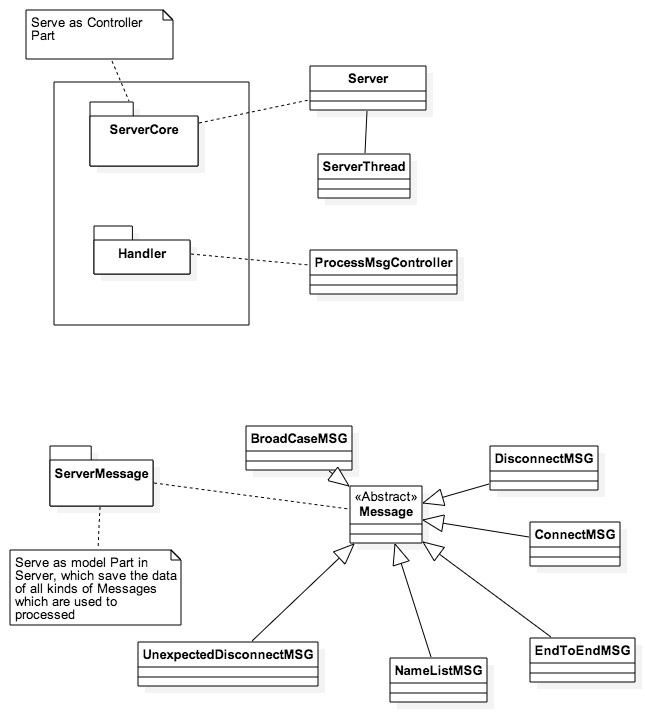
In Controller, we have three classes:

The Server class is responsible for accepting sockets, dispatch threads for different threads to serve clients, removing dead connections and do the clean-up. Server is also a interconnection among clients, to broadcast, send whispering messages between two specific sockets and verify duplicate users.

The ServerThread class is responsible for responding to each request issued by clients, and forward these requests to ProcessMsgController or Server to complete the actions.

The ProcessMsgController will process different requests from user. These messages are transported in socket so it’s flatten. ProcessMsgController help recover these messages and process these messages respectively.

The Class Diagram of server side is displayed as figure below.



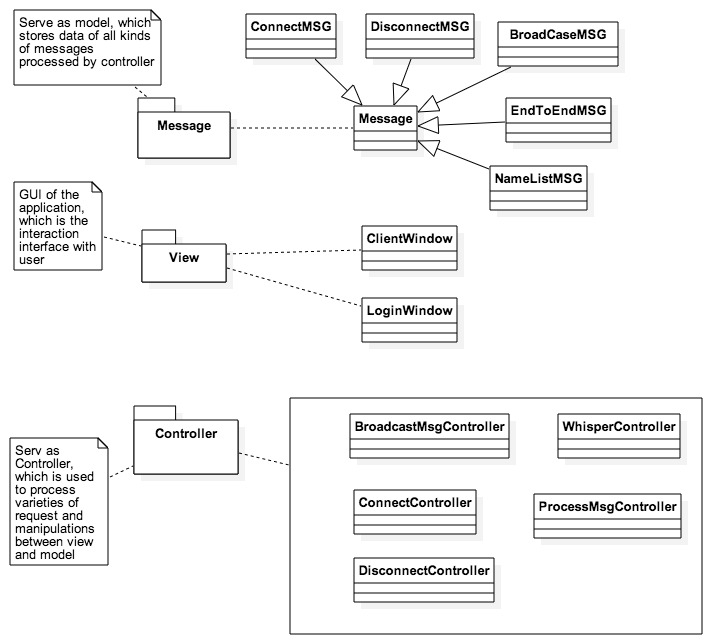
## Client-Side Design Architecture

In model, we have similar messages classes as in Server. Since Server and Client can be separated, the communications between them requires them to comply consistent Message format. So the messages classes are basically the same as in Server.

In Controller, we have:

|  |  |
| --- | --- |
| Class name | Responsibility |
| BroadcastMsgController | the class will create a Broadcast message, flatten this message using a consistent format and write to the outputstream to the socket owed by the sender. |
| ConnectController | the class will create a socket and a connection message, flatten this message which attach the user’s nickname to the outputstream of the socket owed by the sender. |
| DisconnectController | the class will create a disconnect Message and flatten this message which attach the user’s nickname to the outputstream of the socket owed by the sender. |
| ProcessMsgController | The class will parse a flatten message to a real message. And process these messages accordingly. |
| WhisperController | The class will create a Whisper message and ask write the whisper request to the client’s outputstream. |

In View, we have two classes, LoginWindow is used for client to issue connection request, after login, the ClientWindow will be the main GUI for client to interact with the app.



# Testing and Evaluation

## Test Metrics

|  |  |
| --- | --- |
| Server | Server operations (such as connect requests and disconnect requests) should be printed out by the server. |
| The server must handle connections / disconnections without disruption of other services. |
| Clients must have unique nicknames, duplicates must be resolved before allowing a client to be connected. |
| All clients must be informed of changes in the list of connected users. |

|  |  |
| --- | --- |
| Client | A list of online users must be displayed (via GUI or command). |
| Connection / disconnection actions of users must be displayed. |
| Messages from the originating user and other users must be displayed (in other words the messages you send must also be displayed). |
| Must still be able to receive messages / actions while typing a message. |
| Clients must be able to disconnect without disrupting the server. |

|  |  |
| --- | --- |
| Stability and Error Reporting | Updating user list after user connection / disconnection |
| Client Stable after Server Termination |
| Server Stable after Client Termination |
| Whispering to a Non-Existing Client |
| Trying to Connect to Non-Existing Server |

## Server Test cases and results

1. Server operations (such as connect requests and disconnect requests) should be printed out by the server.

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Start server, start two clients, close one of them. | | Two connections messages shown and one disconnection message shown. | Yes |
| result |  | | |

2. The server must handle connections / disconnections without disruption of other services.

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| One client log in, broadcast a message, then another client come in, first disconnect, then second broadcast message | | Connection won’t disrupt the first client, disconnection won’t disrupt the second one | Yes |
| result | first client window before leaving    Second client after fist leaving, continue sending mesage | | |

3. Clients must have unique nicknames, duplicates must be resolved before allowing a client to be connected

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| start two clients, the latter choose the same nickname as the former | | The second client cannot get connected to server and notified with a warning | Yes |
| result |  | | |

4. All clients must be informed of changes in the list of connected users.

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| One client log in, then another client come in, first disconnect, | | First can see the two connection information and consistent connected users  Second cansee her own connection information and disconnection message shows up and list get updated | Yes |
| result | First client:    Second client(After first gone) | | |

## Client Test cases and results

1. A list of online users must be displayed (via GUI or command).

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| start three clients one by one | | Everybody get a list of online users | Yes |
| result |  | | |

2. Connection / disconnection actions of users must be displayed.

Test case and result of first scenario can also be used here.

3. Messages from the originating user and other users must be displayed (in other words the messages you send must also be displayed).

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Three clients connected, they all broadcast a message | | Three clients can see all the three different messages | Yes |
| result |  | | |

4. Clients must be able to disconnect without disrupting the server.

Test case and result of second scenario of Server can also be used here.

5. Whisper to another online client

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Alice and Bob whisper with each other, there is another user Amy | | Alice and Bob can see their whispering messages, however Amy cannot | Yes |
| result | Alice:    Bob:    Amy: | | |

## Stability and Error Reporting Test cases and results

1. Updating user list after user connection / disconnection

Test case and result of fourth scenario of Server can also be used here.

1. Client Stable after Server Termination

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Start server, one client log in, stop server | | client be stable | Yes |
| result | Although client cannot send messages anymore since the server dies, their windows are stay active, no exception, die state appears. | | |

1. Server Stable after Client Termination

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Start server, one client log in, stop the client by force quit it | | Server didn’t get interrupted, it can continue serving other clients | Yes |
| result | Other clients can see the disconnection of that die client, and can continue talk to each others. | | |

1. Whispering to a Non-Existing Client

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Start server, one client log in, want to talk to a non-existing client | | The client get forbidden to do so | Yes |
| result | Since by default, we broadcast message, or we can only choose client which are on the active list, who are all exsting clients, this case never happens. | | |

1. Trying to Connect to Non-Existing Server

|  |  |  |  |
| --- | --- | --- | --- |
| Senerio | | Expected Result | As Expected? |
| Start server, one client try to connect but use a wrong port, which indicates choosing a non-existing server | | The user get a warning and cannot successfully log in | Yes |
| result |  | | |

# Future Development

In the future, there are two aspects I think can be improved.

## about GUI

First, we can beatify the design and improve user experience. For example, we can change the background as a beautiful image, allow users to set their profile and select different fonts/ colors/ letter size to help differentiate other users’ messages and the client its own message.

Second, we can implement multi-windows for Whispering scenarios. Since although integrating whispering messages and broadcast messages are convenient, they sometimes can make clients hard to spot. Fonts/colors may help reduce confusion in some extent, but not a ideal way. Multi-windows are much better.

## About functions:

First, add reconnection function when socket fails to connect to the server or socket suddenly loses connection. This can free clients by stopping letting them reconnect by hand time to time. This is one way to improve user experience.

Second, we can add log in/ user authentication to keep users in records. Adding database support to help log this information.

Third, let client to download or save the history records of chatting.

# Conclusion

This report has describe the successful design and development of development a chat program based on the client-server model. The design and program are modular in nature and make good use of abstract data types and of software re-use. The report includes test cases used to verify the correct operation of the program, as well as the entire code. This project we realize all specifications required in the Project Description Document.

Basically, they are:

1. Muktiple client
2. No GUI is needed for the server
3. A simple GUI can be implemented for the server
4. Clients must be able to “whisper” to each other
5. Clients must be able to choose a nickname

From this project, I learned a lot from this experience. It gives me a deeper understanding about the network and strengthen several kinds of skills: Programming skills, Designing skills, Code Refactoring skills etc. It’s a greate experience to implement this network project.

# Appendix

## Reference

“Building a Java chat server”, ibm.com/developerWorks

## Code

Since the number of code lines of this project is big, it’s not feasible to put all codes here, I post my code on Github, which can be reviewed and downloaded in this address:

<https://github.com/susansan77/Chatroom.git>