David Kim

Term Project: *Java Messenger*

Design Document

Table of Contents

1. Introduction

This document describes in detail the software requirements for Java Messenger, a simple online instant messenger program. This document will describe the problems Java Messenger intends to address and the functional/non-functional design requirements of the proposed system. This document is intended for the stakeholders of the application and to assist in the development process of Java Messenger as well as serve as a reference to clarify any future issues the stakeholders may encounter.

1.1 Purpose and Scope

The purpose of this Software Requirements Document is to layout the foundation for Java Messenger, a chat application that consists of a chat server and an indefinite number of chat clients (users). The following developer-oriented requirements describe the system from a software developer's perspective. These requirements include a detailed description of functional, data, performance and other important project requirements.

1.2 Target Audience

The target audience is a company or corporation wishing to facilitate better communication and productivity. Each employee will have their own private account and be able to send and receive messages thereby facilitating inter-corporation communication while ensuring a robust secure network for company privacy.

1.3 Terms and Definitions

Term	Definition
User	Someone who interacts with the messaging service
Stakeholder	Any person who has interaction with the system who is not a developer
Redundancy	The duplication of critical components or functions of a system with the intention of increasing reliability of the system, usually in the form of a backup or fail-safe, or to improve actual system performance
Server	A computer or computer program that manages access to a centralized resource or service in a network
GUI	The graphical user interface is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.
SDLC	Software Development Life Cycle
Socket	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 destined to be sent to. An endpoint is a combination of an IP address and a port number.
Backend	Backend refers to the parts that do the work behind-the-scenes while the user is unaware of or cannot see. This includes databases, services, etc.

2. Design Considerations

This section describes constraints and dependencies of the system as well as the methodology used to design the system.

2.1 Constraints and Dependencies

The project's constraints include the time requirement for the project and resources

necessary for development. The project's final deadline including the project report and final deliverables are due no later than June 8th, 2017. It is expected that I have a functional application along with the required documentation by the due date. The deadline limits the time to develop the test plan to 3 weeks and time to prepare the project report and final deliverables to 1 week.

The following are some of the dependencies within the project. The execution of the test plan requires that the first version of the application is completed and semi-functional. Before the final product can be delivered, all code will be tested and all supplemental documentation should be updated according to the testing results.

2.2 Methodology

An interactive software development life cycle, or SDLC, model known as the Unified Modeling Language was used to develop Java Messenger. By modeling the system after the client/server architecture, I am able to compartmentalize the application into several logical subsystems. This results in a well-defined, modularized design.

3. System Overview

The system developed for this project allows multiple users to send and receive instant messages through a client application and a chat server. The client application allows the user to view and manage a contact list and start/end conversations with other users registered within the system.

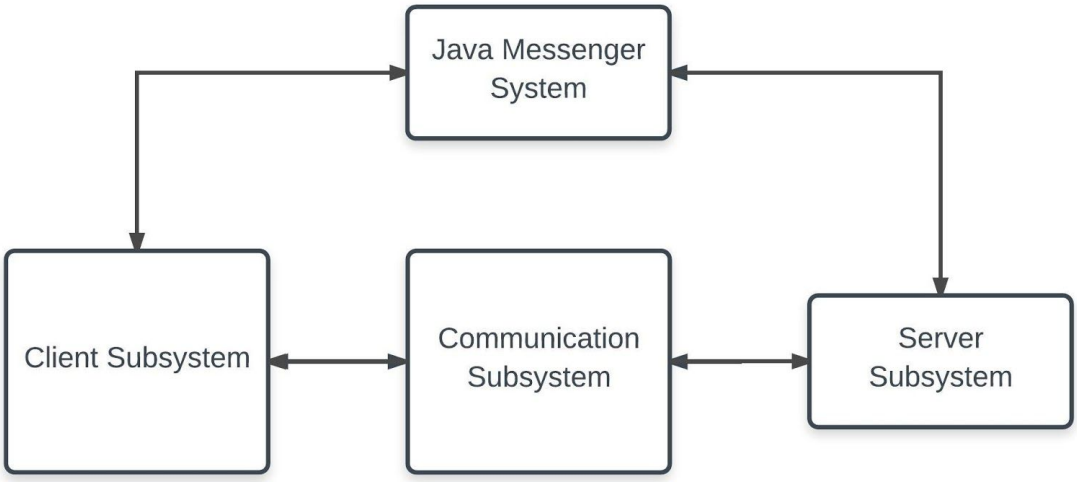

Figure 1: System and Subsystems Flowchart

4. System Architecture

System architecture (or Software architecture) to which the system design must conform to is an important step in the development of any system. This section provides the clear and concise criteria to guide the development of the project. The following subsections describes the Client, Server and Communication subsystems and their respective subcomponents.

4.1 Client Subsystem

The client subsystem contains all the components related to the GUI/Windows on the client end. The subsystem monitors the communication link and manages the communication packets (which are generated by the communication subsystem). These packets are sent to the server subsystem. The client subsystem also maintains, stores and updates all local data contained within the subsystem. The client subsystem is split into the following four primary classes: JMClient, LoginWindow, ContactListWindow and ChatWindow.

4.1.1 JMClient Class

The JMClient class is at the top within the Client subsystem hierarchy. It creates windows that are used by the user to interact with the system. JMClient creates the socket connection with the remote server. It also manages communication between the user, the multiple windows and the server.

4.1.2 LoginWindow Class

The LoginWindow class is the main entry point as it opens the login window upon startup. The window allows the user to enter in a name and password, or register a new account. The authentication process or registration process is handled by the backend. After successful authentication or user registration, the ContactListWindow will appear.

4.1.3 ContactListWindow Class

When the user wants to initiate a chat they can simply double click on a contact's name within the user's contact list. Then a new ChatWindow will appear which allows the user to send and receive messages.

4.1.4 ChatWindow Class

The ChatWindow class manages incoming and outgoing messages between the Client application and the server. The user may type a message in the send message box located on the bottom of the chat window. Then the user may press enter or click on the send button to transmit the message. Any incoming messages received will be displayed in the receive message box located above the send message box within the chat window.

4.2 Server Subsystem

The server subsystem manages all the connections between the server and its

clients. The system processes message routing between clients. It also manages stored data such as chat history and user passwords.

4.2.1 JMServer Class

The JMServer class is at the top within the Server subsystem hierarchy. It initially opens a socket and begins listening for a remote connection. When a remote connection is established, a new thread is generated to handle the new client. The server then continues to listen for future connections.

4.2.2 Handler Class

The Handler class works behind the scene to ensure proper functioning of the system. It manages the authentication process, clients contact lists, and routes messages to proper recipients.

4.2.3 UserList Class

The UserList class manages a list of all the users currently logged into the system.

4.2.4 User Class

The User class creates objects which represent the users logged into the system.

4.3 Communication Subsystem

The communication subsystem contains the command interface and protocol used by the client and server subsystems in order to communicate messages between each subsystem.

4.3.1 Command Class

The Command class represents the main interface for the communication

subsystem.

4.3.2 MessageCommand Class

The MessageCommand class represents the message interface.

4.3.3 ContactListCommand Class

The ContactListCommand class manages the list of logged-in users within a client's contact list.

4.3.4 UserStatusCommand Class

The UserStatusCommand class maintains the current statuses of each user within a client's contact list.

4.3.5 LoginCommand Class

The LoginCommand class facilitates the authentication process by username and password.

4.3.6 ErrorCommand Class

The Error Command class is used to handle program/user errors and exceptions.

4.3.7 LoginErrorCommand Class

The LoginErrorCommand class is used to handle notification for authentication failure.

4.3.8 Message Class

The Message class generates the message object along with the source, destination and message data.

5. Detailed System Design

This section describes in detail the system design along with its various subsystems and subcomponents. The following subsections describes in detail the Client, Server and Communication subsystems and their respective subcomponents.

5.1 Client Subsystem

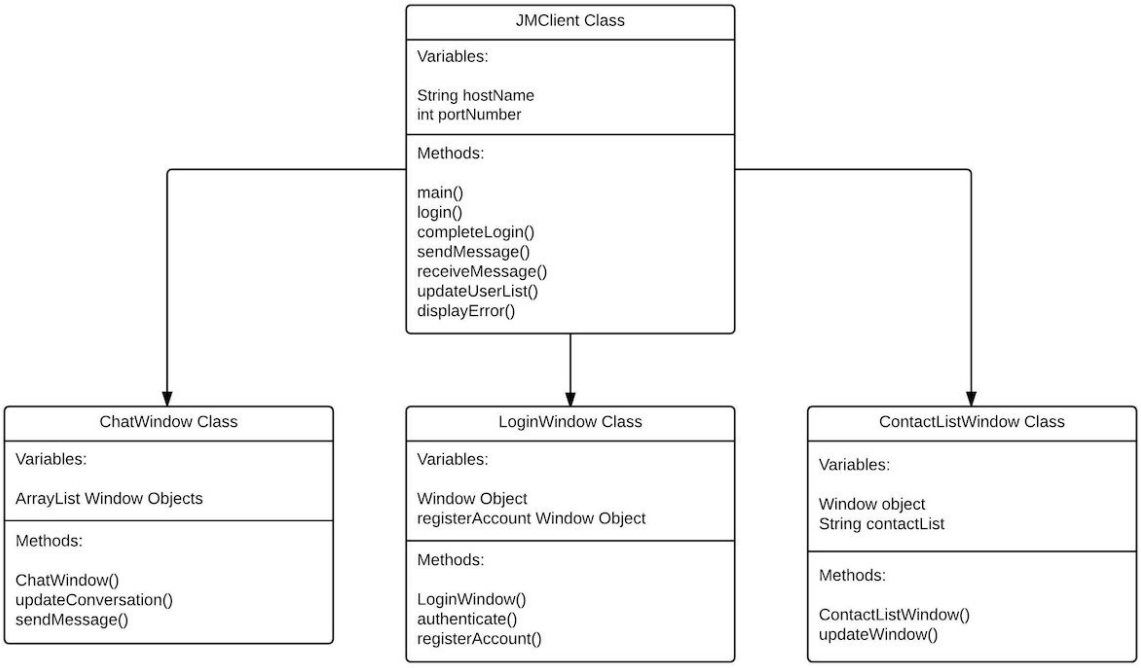

Figure 2: JMClient Class Diagram

5.1.1 JMClient Class

This class will contain the following methods: main(), login(), completeLogin(), sendMessage(), receiveMessage(), updateUserList() and displayError(). The main() method is executed at start up and begins by creating a socket connection to the server. After successful connection is made, the method will create a new instance of a LoginWindow in a separate thread of execution. Then, the client waits in a loop and processes

commands from the server. The login() method authenticates a user by checking with the server. The sendMessage() method sends a message and the receiveMessage() method handles incoming message packets by displaying them on-screen. The updateUserList() method updates the status of users in the contact list.

5.1.2 LoginWindow Class

The constructor LoginWindow() opens a login window. The login window is prompted to enter in a username and password. After clicking on the "Login" button, the authenticate() sends the username and password to the server for authentication. The user may also sign up a new account by clicking on the "New User" button. Then, the registerAccount() method opens a window that handles a new account registration.

5.1.3 ContactListWindow Class

The constructor ContactListWindow() opens a window that contains the user's contact list which contain the following: online users and user's friends. By double clicking on any user in the contact list window, the user can initiate a conversation. This method also listens for commands from the server which notify the window that a chat has been initiated.

5.1.4 ChatWindow Class

The constructor ChatWindow() opens two windows in separate threads of execution. The top window displays the entire conversation between clients and the bottom window is where the client types out their message. When typing is completed, the user presses the "Send" button. This executes the sendMessage() method creates and sends a message packet to the server. Subsequently, the updateConversation() method is executed which updates the conversation in the chat window.

5.2 Server Subsystem

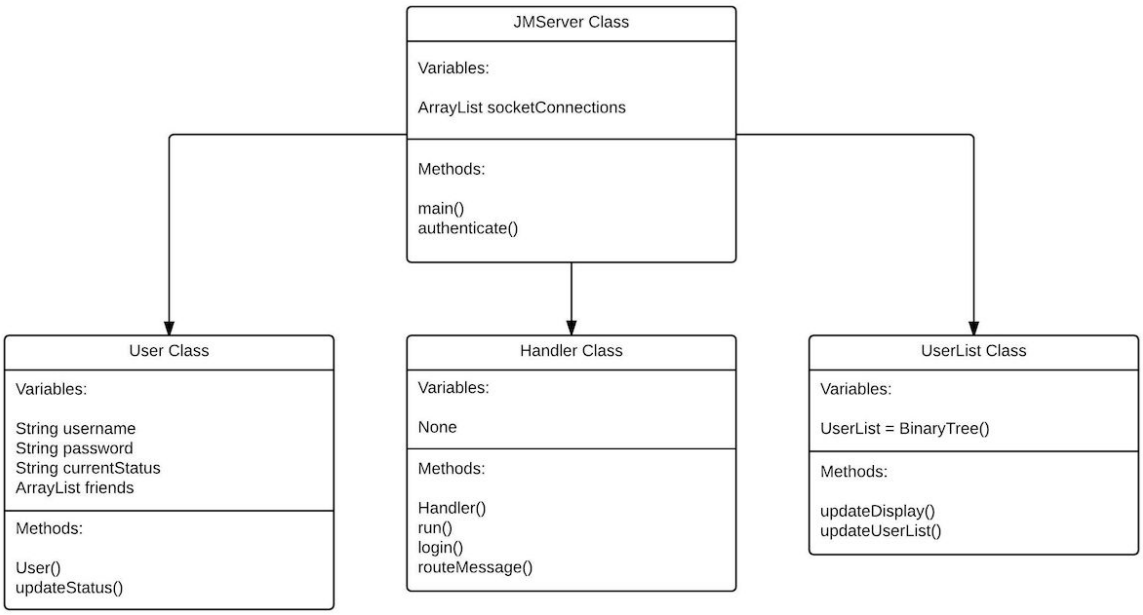

Figure 3: JMServer Class Diagram

5.2.1 JMServer Class

This class will contain the following methods: main() and authenticate(). The main() method is used to initiate the server. It begins by generating a socket to which a remote client can connect. When a connection is created, the connection is passed to a new thread handler and the server continues to listen for the next connection. The authenticate() method authenticates a user that is attempting to log on.

5.2.2 Handler Class

This class extends Thread and contains the following methods: Handler(), run(), login(), routeMessage(). The constructor Handler() handles a new socket and JMServer object. The run() method opens input and output streams for use. The login() method attempts to login a user. The routeMessage() method performs the necessary message routing.

5.2.3 UserList Class

This class handles User objects in a sorted binary tree. All necessary binary tree operations will be included within this class.

5.2.4 User Class

The constructor User() initializes a User object, which includes the username and password, the user's current status, an ArrayList of friends for the user. All necessary ArrayList operations will be included within this class.

5.3 Communication Subsystem

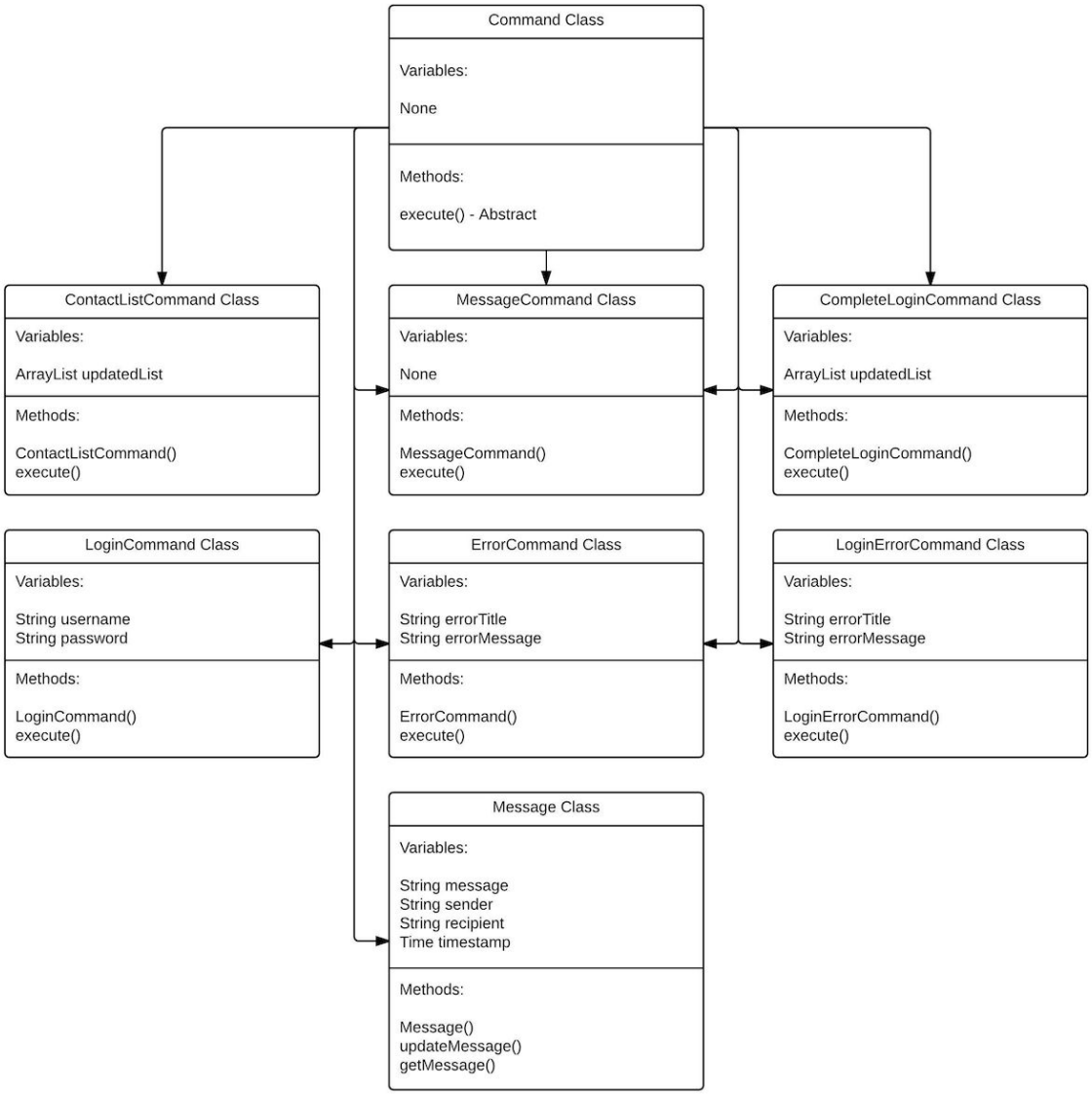

Figure 4: Command Class Diagram

5.3.1 Command Class

This interface class defines the following abstract method: the execute() method within this class directs the executing command object to one of the various classes within the communication subsystem. The accepting class properly defines execute() for the specified objects.

5.3.2 MessageCommand Class

The constructor MessageCommand() creates a new object, which consists of a message object. This class sends the message packet either to the client application or the server. The execute() method initiates the sending of messages.

5.3.3 CompleteLoginCommand Class

The class constructor CompleteLoginCommand() creates a new object that holds an ArrayList of all the user's friends that are the logged on. The execute() method will send an update request to the client application.

5.3.4 ContactListCommand Class

The class constructor ContactListCommand() creates a new object that holds an ArrayList of the names of user's friends logged on and the user's current state. This class updates the contact list using execute() method.

5.3.5 LoginCommand Class

The class constructor LoginCommand() creates a new object, which consists of the username and password. This class executes the LoginCommand commands by using its execute() method.

5.3.6 ErrorCommand Class

The class constructor ErrorCommand() creates a new object, which consists of the message to be displayed, and the title of the box. This class executes ErrorCommand commands by using its execute() method.

5.3.7 LoginErrorCommand Class

The class constructor LoginErrorCommand() creates a new object, which

consists of the message to be displayed, and the title of the box. This class is a specific type of ErrorCommand. This class executes LoginErrorCommand commands by using its execute() method.

5.3.8 Message Class

The constructor Message() creates a new object, which consists of a destination (recipient), source (sender) and message. There are several methods defined which can access each data member.