Bradley Maness

Term Project: Java Chat App

Design Document

**Table of Contents**

**1 Introduction** **2**

1.1 Purpose and Scope 2

1.2 Target Audience 2

**2 Design Considerations\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3**

2.1 Constraints and Dependencies\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

2.2 Methodologies\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3

**3 System Overview\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4**

**4 System Architecture\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6**

4.1 The Client Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

4.1.1 UserInfo Class Object\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

4.1.2 Convos Class Object\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

4.1.3 History Class Object\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

4.1.4 OnlineUsers Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

4.1.5 Other Client Class Data Members\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_7

4.1.6 Client Class Methods\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_8

4.2 The Server Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

4.2.1 Connections Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

4.2.2 UserData Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

4.2.3 Other Server Class Data Members\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_9

4.2.4 Server Class Methods\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

**5 Detailed System Design\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11**

5.1 Client Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

5.1.1 Thread Class and Message Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

5.2 Server Class\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

5.2.1: Saving and Loading User Data and Message History\_\_\_\_\_\_\_\_\_\_\_\_\_\_11

**1 Introduction**

**1.1 Purpose and Scope**

The purpose of this document is to provide a detailed overview of the project’s design and structure. This document will explain the main workings of the project’s two aspects: the Client and Server classes. It will provide an overview of the class objects, their data types (primitive and abstract) and methods, and explain their functionality and interconnectedness to each other in regards to the project.

**1.2 Target Audience**

The target audience for this document consists of the instructor and their assistants for the purposes of demonstrating techniques for documenting software design for scholarly purposes, as well as for more practical reasons regarding anyone who wishes to read general design documentation on a simple product.

**1.3 Terms and Definitions**

This paper is written in the context of the Java programming language. Generalized terms will be kept to a minimum.

Class: a type of file in Java that consists of data members and methods.

Object: an implementation of a class.

**2 Design Considerations**

**2.1 Constraints and Dependencies**

For this project the only constraints given are those that are inherent in what the java programming language can do, such as the abilities of the specific libraries like java.swing (GUI programming) and java.net(network programming).

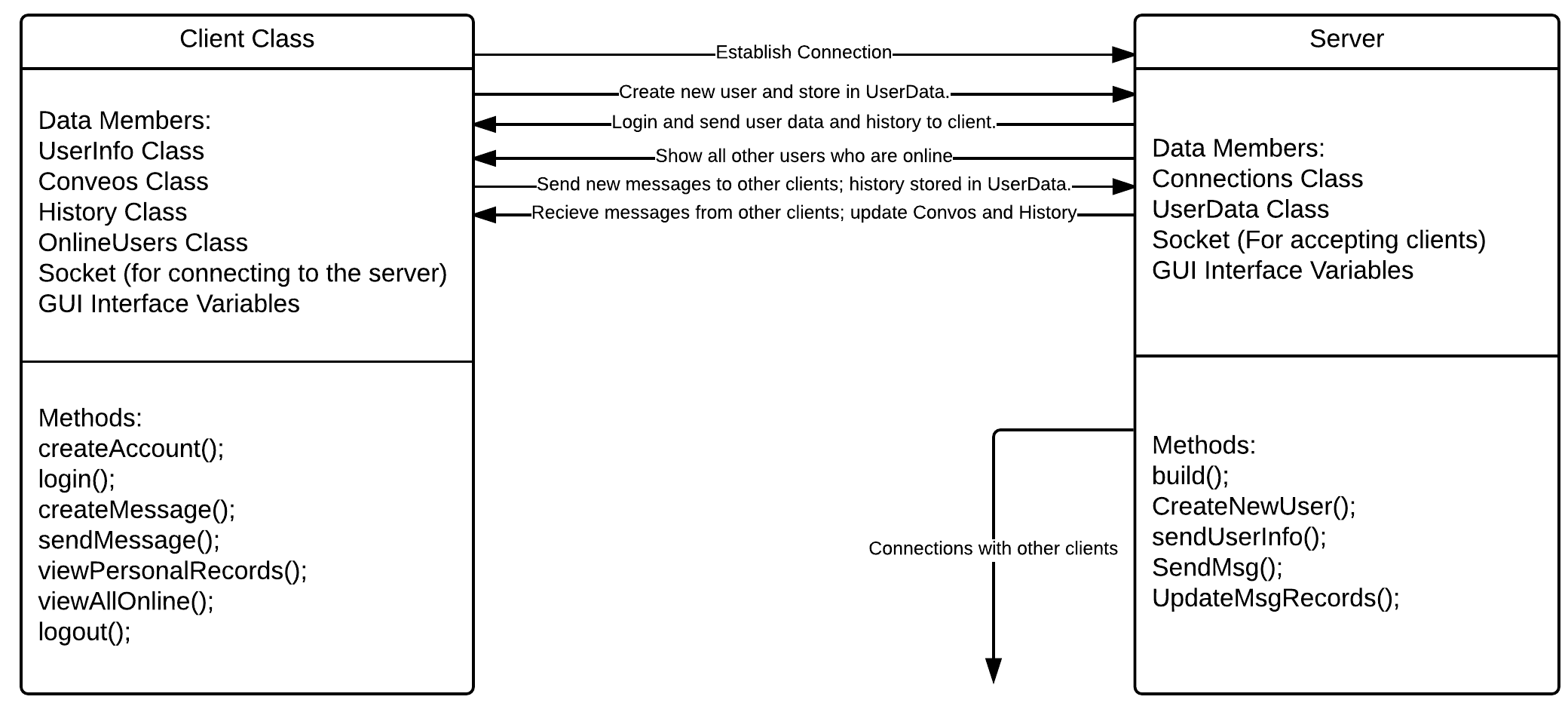
**2.2 Methodology**

This project does not strictly adhere to one form of software engineering methodology. The best way to describe it would be a mix of the Object Oriented Design(private data members with member methods to interact with them illustrated with class charts) and Incremental Development(Smaller scale waterfall processes in which sections of the project are constantly being tested and refined).

The contents of this paper should not be considered as a final representation of the project. This paper is rather a reflection of the project at it’s most current iteration and is subject to change should the project require it.

**3 System Overview**

The basic system of this project is communication between two different programs(the client(s) and the server) in order to deliver messages between different machines. This section will cover the basic abstraction of the projects main components, their sub-components, as well as the important relationships between them using graphs and figures wherever necessary.

Figure 3.1: Basic overview of Client-Server System.

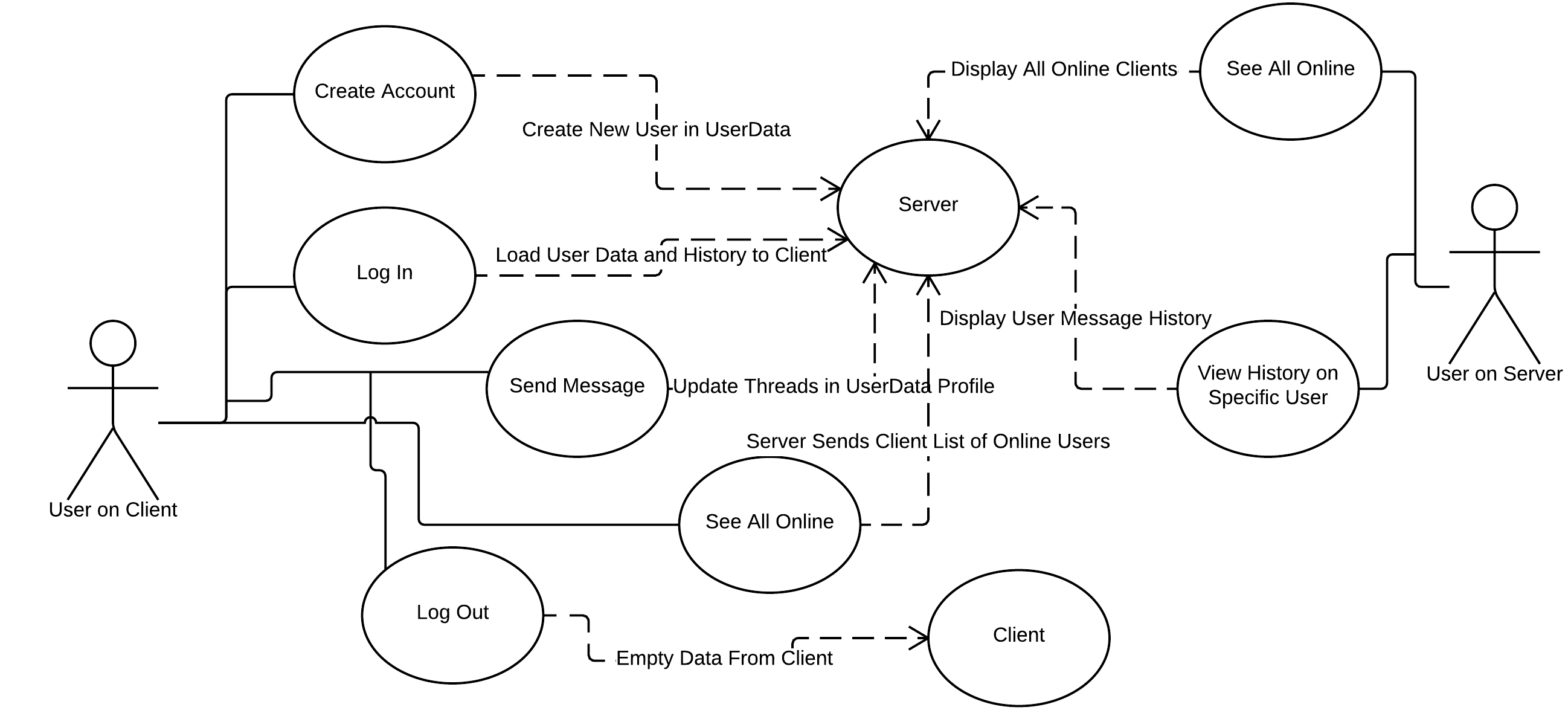


Figure 3.2: Basic Use-Case Diagram for people using Client and Server Classes

The previous figures show the basic relationship between the two main classes of the project, the Client and Server classes, and the basic use-cases for both. Each class is an amalgamation of other class objects that are designed to handle each of the project’s specific requirements using abstract data types. Further explanations for each component and subcomponent will be provided in sections 4 and 5 respectively.

**4 System Architecture**

This section will describe the workings of the first layer of the overall system that users will be working with: the client and the server classes It will provide a brief overview of each class’s subcomponents and provide a summary of their significance and functionality. Further elaboration on each subcomponent’s functionality will be provided in section 5.

**4.1 The Client Class**

The Client class will be the main object in the system that users will spend the most time interacting with. The client will provide a basic interface from which to send, receive, and view messages from via the Server class. Firstly, the Client class must include variables that set up a graphic user interface as well as connect to the Server class via socket variables in the java.net library. Besides these variables, the Client will also be an amalgamation of several other class objects that will be responsible for handling the functional requirements of the product.

**4.1.1 UserInfo Class Object**

This component will be responsible for containing user information. In the context of the Client class object it will be used to store data of the user currently logged in. It will contain the the user’s name (real name), username (name to be displayed for messages), and password. This information is set when the user successfully logs in and is deleted when the user logs out. This class will also be used as part of the user information keeping in the Server class.

Necessary methods this Object must include:

A constructor to set all variables to specific values:

void UserInfo(String name, String usrNm, String pswd);

A copy constructor for copying objects when needed:

void UserInfo(UserInfo);

“getters” for retrieving data whenever needed:

String getName();/getUser();/getPassword();

**4.1.2 Convos Class Object**

The Convos class (shortened from “Conversations”) is a class object member of Client that provides a subsystem for handling messages received from the Server class. It will use a doubly linked list data structure to store and manage a list of Thread Class objects (To be explained in section 5) which contain ongoing messages between the current user and other users on the server. The messages will be organized in chronological order with threads that contain the most recent message by date at the beginning of the list. A doubly-linked list was chosen so that the list of conversations the user takes part in can grow or shrink in size whenever needed.

The Convos Class must contain two Thread Class objects as data members to contain the list with one acting as the head (most recently updated) and the other as the end(oldest conversation).

As method members, Convos must include all required methods of a doubly-linked list (insert(), delete(), search(), display()) as well as a special method for organizing the list:

Update(); add a new message to a specific thread and move that thread to the beginning of the list;

**4.1.3 History Class Object**

The History class object is another subsystem within the Client class object that contains a user’s entire conversation history. This history will be loaded in from the Server Class’s ConvoHistory Class object when the user logs in. The history is a collection of Thread Class objects stored in a binary search tree in chronological order of when the threads were created. Within this data structure the user will have the option to add specific threads to their Convo Class object if they wish to continue conversations instead of starting new ones. This class object will also be constantly updated whenever a user sends or receives a new message. When the user logs out, the data structure will become empty.

As a binary search tree this class object must have one Thread class object as it’s root as well as basic BST methods for insertion, searching, and deletion. In order to add Thread class objects to the Convo class it must contain a special method for returning a Thread object to add such as “Thread addToConvo()”.

**4.1.4 OnlineUsers Class**

This class will be used to display a list of all other active users who are currently logged into the server. The class object will store this data in a dynamic array of Strings provided by the server and display all active users in a list. When a user logs out the server will “pulse” and send a new list of online users in order to update all connected Client’s OnlineUsers lists.

Data members for this class include a dynamic array a Strings of which size will be automatically determined when the user logs in.

Method members must also include a function for updating the list as well as the initial constructor.

**4.1.5 Other Client Class Data Members**

For connecting to the server the client will have a Socket class object provided from the java.net library as well as readers and writers from the java.io library for sending and receiving data and objects between the Client and Server class objects.

The GUI interface will be constructed after main functionality has been implemented using the java.swing library. It will consist of a single window of undetermined size with partitions that allow the display of current conversations and other online users as well as allow the user to look up conversation history and add it to current conversations. It will also contain a bar for alerts when a new message is received from the server.

**4.1.6 Client Class Methods**

The Client class must have methods that allow the following actions:

createAccount(); Allow unregistered users to create a new account to

be added to the server.

Login(); allow existing users to connect to the server.

CreateMessage(); allow logged in users to create messages, address

them to other users and send them to the server.

ViewAllOnline(); allow logged in users to see all other users currently logged in.

viewPersonalRecords(); allow logged in users to view their complete

history of messages they are included in.

logout(); logs the user out and removes existing data from all other

class objects within the Client class.

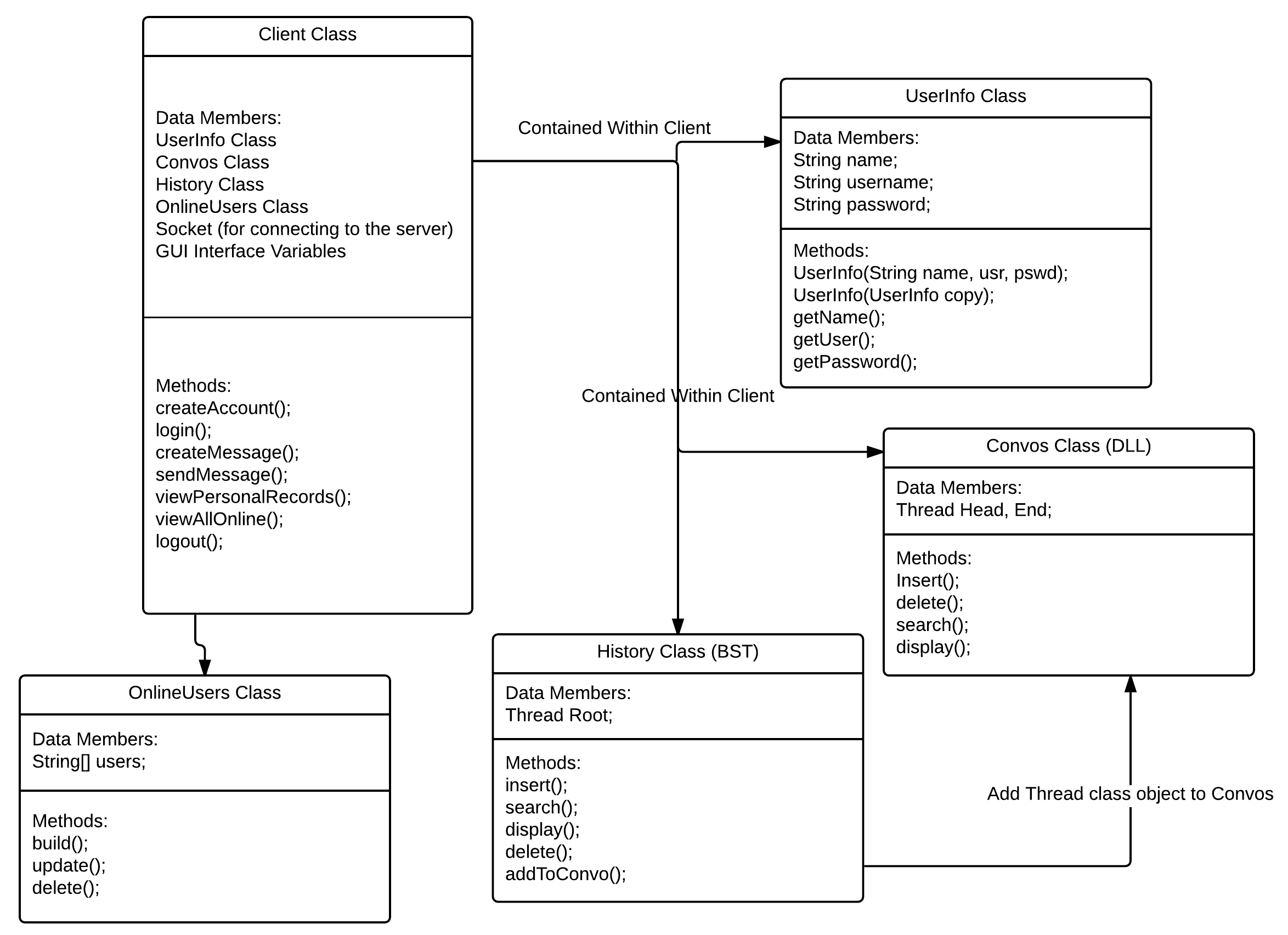


Figure 4.1: Overview of all Client subclasses

**4.2 The Server Class**

The Server class object is the second greatest object within the project. It will be responsible for handling connections from multiple Client class objects running on different machines as well as storing user data, passing messages between connected users, and keeping a history of all messages passed through it. To accomplish this, the Server, like the Client, will be an amalgamation of multiple class objects as well as primitive variables.

**4.2.1 Connections Class**

The Server must be able to handle an indefinite number of users as per the requirements. The Connections Class is an object designed to accept connections from Client objects and create separate threads for each connection. The Connections Class will also need to handle requests from Clients as well as pass info between different Clients and the other Server Objects. This class will use other classes contained in the java.net library to accomplish it’s tasks.

**4.2.2 UserData Class**

The Server must be able to store both basic user information as well as conversation history for each user. The UserData Class will be created to accomplish this. This class will be implemented as a binary search tree data structure organized by name(real name) and will contain another class object called Records which will references to a UserInfo class object(Client 4.1.1), a History class object(Client 4.1.3), a boolean value to check if the user is logged in, as well as left and right Record references for BST node functionality. When a user connects from a client and logs in the tree is to be traversed to find a matching name and password to be returned to the Client class. When a message is sent between different Clients, the tree will be traversed and all users involved in the conversation will have their histories updated with that message.

Because the Server class must store data permanently it will require a subsystem to write data to and read from a file system containing data on all users as well as all conversations by those users. This system will be described in more detail in section 5.

Because data is being kept in a binary search tree it will require a single node reference to act as the root as well as functions for insertion and searching. A primitive Boolean variable will also be added to the node to show whether or not the user is online.

**4.2.3 Other Server Class Data Members**

The Server class requires data members for providing networking functionality as well as generating a user interface. It will require a ServerSocket Class object from the java.net library to accept incoming connections and an int primitive to provide a port number for Client connections. As with the Client class the user interface will be created using the java.swift library after all functionality is achieved. It will provide a window to see all users who are online and allow the person who set up the server to view data on other users as well as all message records for all users.

**4.2.4 Server Class Methods**

The Server class must implement these methods to achieve functionality:

build(); When a Server class object is created the constructor must

read in user data to the respective class members wherever applicable.

CreateNewUser(); create and store a new user profile when a new,

unregistered user requests it.

sendUserInfo(); send specific user info to a connected Client when

proper login info is provided.

SendMsg(); when a connected Client sends a message to the server,

it must send it to all other connected users addressed in that message.

UpdateMsgRecords(); whenever a message is sent across the server, all

the message histories of the sender and addressees must be updated

with the message.

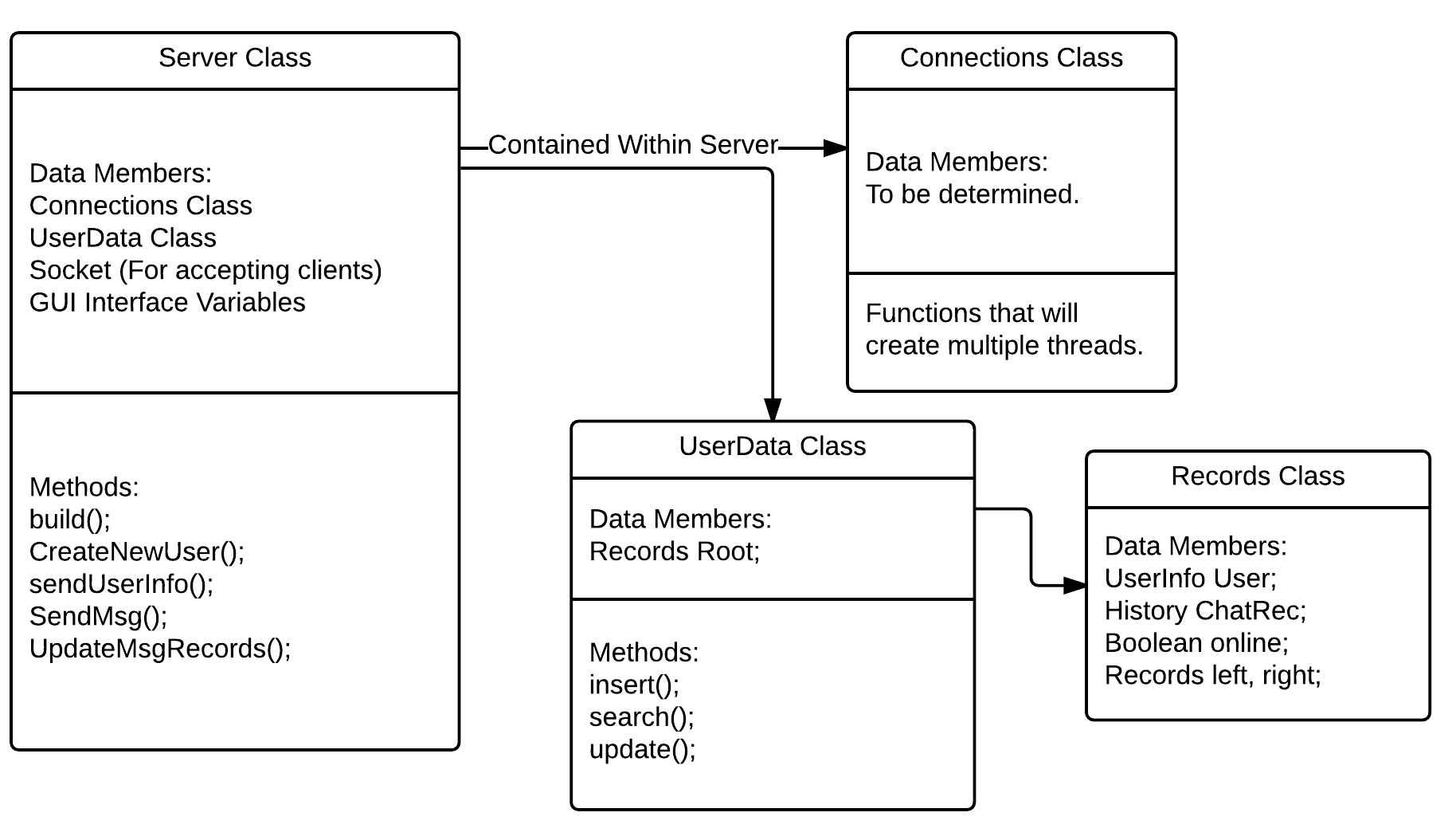


Figure 4.2: Overview of server subclasses.

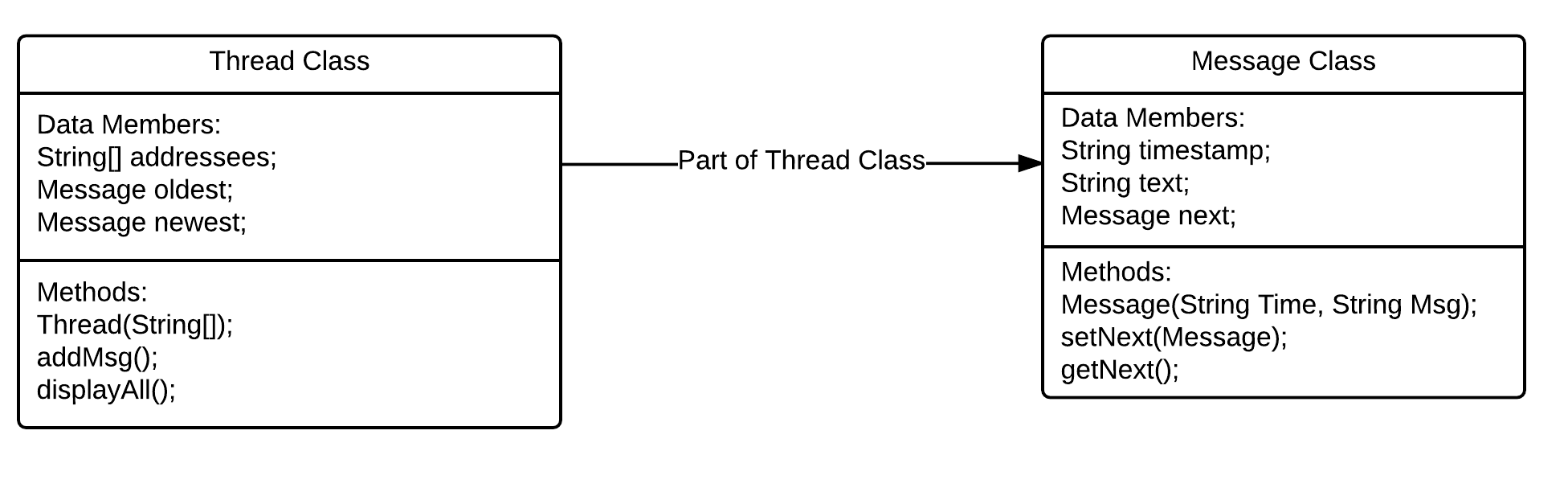
**5 Detailed System Design**

This section will cover the functionality of subsystems contained deeper within the Client and Server classes.

**5.1 Client Class**

**5.1.1 Thread Class and Message Class**

The Thread class is a subclass contained within the Client’s Convos and History member classes. The purpose of the Thread class is to contain a list of addressees as well as manage a linear-linked list of Message class items. In any thread, only those listed as addressees are allowed to add Message class objects to a respective thread.

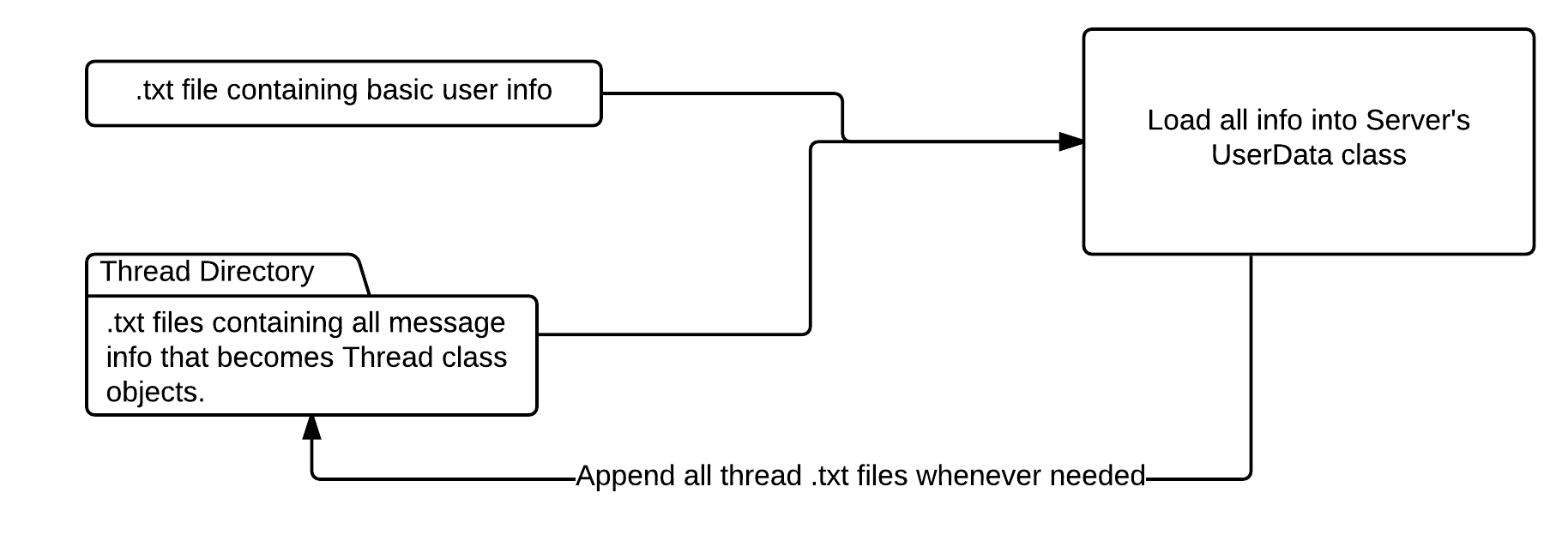
Figure 5.1: Thread and Message Subclasses

When a new thread is created it is initialized with an array of Strings that contains a list of any number of addressees. When a message is sent to the server it is then delivered to those on the addressee list, including updating their server history. Messages are organized newest first and oldest last. Each message consists of a string and a timestamp of when it was sent.

**5.2 Server Class**

**5.2.1: Saving and Loading User Data and Message History**

Whenever a Server class object is created it must build its user database and message history from external saved file. While running it must also update all files or create new ones wherever needed. Because data for both users and their respective message histories are being kept in the same BST node for the Server’s UserData class the server will read and write to directories corresponding to unique users. For example: user “John Smith” will have his own directory bearing his name. Within that directory will be a .txt file that contains the info that goes into the UserInfo class. There will also be directory containing .txt files that correspond to each thread that the user is involved in. When a new message is written to a thread the server will update .

Figure 5.2: Whenever a new message is sent across the server, for each user involved, a new file is created for the thread or an existing thread file is updated.