Project Report

Name: David Kim **PSU_ID:** 948758537 **Email:** djk3@pdx.edu

Java Messenger is a simple online instant messaging service that consists of a chat server and multiple chat clients called users. It is a tool used to communicate with other people over the Internet in real time. The program, written in Java, includes the following features:

- Java Messenger includes a user-friendly GUI which enables users to have conversations.
- New users may register new chat accounts with a username and password.
- Existing users may login using a registered username and password.
- Chat history is saved for future viewing.
- Chat history, username and passwords are stored on the server's database.
- Each user has the option of sending both public and private messages.

Java Messenger was designed to have two main classes: JMClient and JMServer. Each class is self-contained within their own packages, and communication between them is facilitated by Message objects. Each message object contains a type, sender, message content, recipient and time stamp. Each class has a message handler method that effectively handles incoming messages while delegating tasks based on the message type. For example, the JMClient might send a login type message to the server. JMServer's handler recognizes the message type, tries to validate the client's username and password within the user database, and upon successful login, sends a confirmation message back to the client. JMClient's handler will then recognize the confirmation message and update the JMClient GUI as necessary.

The main challenge I faced when designing the chat service was determining the best way to self-contain each class. This is because each application is expected to run on different machines. Therefore, each subsystem requires a certain level of autonomous functionality and encapsulation while maintaining proper channels of communication. Both packages begin with a GUI environment. Each GUI contains objects that represent a main class, sockets, GUI elements, and IP/Port information as well as database and history file path information. In order to better manage runtime capability, I designed each client/server instance to be self-contained within its own thread. Once a thread has started, a listener method, or handler, continues running so that incoming messages can be received on either the client or server application. However, because of the possibility of multiple simultaneous incoming messages, the server's handler employs synchronization which prevents thread interference and memory consistency errors. In other words, incoming messages are sequentially handled in an organized way. This means that one thread can't read or write while another thread is accessing the method.

Another challenge when developing Java Messenger was making sure the GUI elements properly reflected modifications to the controller that handles the inner workings of the program. Proper exception handling was applied throughout the project to ensure that the program wouldn't crash if the client program couldn't reach the server, or the user input was erroneous. Also, I had to decide on a particular XML format for the user database and chat history files. It should be noted that data encryption feature was not included in this build.