Michelle Duer

[mduer@pdx.edu](mailto:mduer@pdx.edu)

ID: 931555117

May 8, 2017

*Babble Chat*

Project Report

This document gives an overview of how to run Babble Chat and details the successfully implemented features and problems that were encountered (as well as solutions or workarounds for the latter). All of the code and documentation is available in my Git Repository.

To run Babble Chat, go to <https://github.com/michelleduer/cs300-project> and clone the repository. Ensure you have the correct libraries installed for testing (hamcrest-core-1.3.jar, junit.jar, junit-4.12.jar), the database (mysql-connector-java-5.1.42-bin.jar), and netty (netty-all-4.1.9.Final.jar) available on the netty website. Others may already be available in your IDE’s library directory. The main methods are found in the Server class, which must be compiled and run in order to get a network created where clients can connect. Next, compile and run one or more instances of the User class. I personally used Intellij IDEA to run and compile the application. You will have to register a unique account to access the chat room.

The biggest challenge with setting up the Netty Server and User classes was getting enough of an understanding to implement the network. This included research, Netty tutorials, downloading the netty-all library, linking the correct library and figuring out how to get the clients to speak to one another (a frustrating hiccup that was resolved by using the *writeAndFush* method rather than the *write*). When running the Server and User, all users are alerted to new logins and logouts by other Users. Netty scales quickly and smoothly from one-to-one communication up to one-to-many communication – this wasn’t implemented perfectly, but the communication in all scenarios does work.

The graphical user interface for the login, registration and chat windows were created using the javax.swing library. I thought I would have only one Interface class, but realized multiple windows would make things easier and ended up creating an abstract base class, Window, to better manage the different interfaces. The Chat window has a log-out button that exits the entire application.

A database (DB) was created to manage a unique id, username and password values for every user. I tried implementing Arch Derby as it seemed to be a prominent Java database, but struggled to incorporate it into my IntelliJ IDE. I switched to MySQL, installed the DB Navigator plugin, necessary mysql library, and easily created the DB through the command line. Creating a successful connection with the database was hard at first, and formatting strings correctly so the sql command required a lot of attention to detail and was successfully implemented.

The JUnit framework available within Intellij was used to create Unit Tests testing for a successful server session, user session, login verification, valid registration, and a successful database connection test.

My main obstacle was integrating the network and graphical user interface. While this was forseen and stated during the Design process, I was unable to fully implement the messaging system via the GUI, although messages are partially integrated and do appear within the interface to alert about logins/logouts, and from the unintegrated instances of the User class, messages will send to the gui Chat window.

If I were to start the project from the beginning again, I would likely use the Java sockets library to more easily implement the messaging system in the short time, but given more time, I would really like to understand Netty’s framework in more depth, create an improved design and refactor my code to better incorporate the interface and network.

I believe there are also some fast workarounds that I created that would need to be resolved. I created a few static volatile variables in the Window class in order to allow login to the chatroom (in the User class) using a thread, which is likely hard on a CPU that is running the program, so not best practice. I began reading up on RxJava, which seems like it would be a safer and more efficient use of threads, but there was not enough time to figure out how to use this instead.

**Overall features and project requirements that were implemented:**

* user registration
* user login/logout
* notifications for logins/logouts
* one-to-one communication
* one-to-many communication
* unit testing
* version control: git
* project report