CSCI 4311 Programming Assignment 1

Christian Fuller

This report serves as a brief overview of the implementation and design of the group chat application.

The application is implemented with the following files:

- ☐ ChatServer.java — *ChatServer and ServerThread classes*
- ☐ ChatClient.java — *ChatClient class*

ChatServer

The ChatServer class's main job is the creation of new ServerThread objects, which each run on their own threads and manage IO for a single client. It also manages the initial setup for the server socket. It contains main().

Main method and creation of the server's ServerSocket object at the port specified by the command line argument:

```
public static void main(String[] args) throws IOException {
    int port = Integer.parseInt(args[0]);
    ChatServer server = new ChatServer(port);
    System.out.println("Server Socket created : " + server.serverSocket);
    server.start();
}

public ChatServer(int port) throws IOException {
    this.serverSocket = new ServerSocket(port);
}
```

The ChatServer class maintains a list of all active ServerThreads so that individual ServerThreads can reference and output to other ServerThreads:

```
private static final ArrayList<ServerThread> threads
        = new ArrayList<>();
```

This is the main loop: accept new client connections and create serverThread objects with that socket :

```
public void start(){
    System.out.println("Server running... accepting clients");
    while (!serverSocket.isClosed()) {
        try {
            Socket socket = serverSocket.accept();
            ServerThread serverThread = new ServerThread(socket);
            Thread thread = new Thread(serverThread);
            thread.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

ServerThread

The ServerThread class is an inner private class that manages IO for its client and relays messages to clients of other ServerThreads. It also handles the creation and closing of streams, and the closing of its socket.

ServerThread constructor: open streams and add reference to current ServerThread to the ChatServer's list of active threads. Streams are set up as: Byte stream -> Character stream -> Buffered character stream :

```
    public ServerThread(Socket socket)  {
        try {
            this.socket = socket;
            this.out = new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream()));
            this.in = new BufferedReader(
                    new InputStreamReader(socket.getInputStream()));
            threads.add(this); // add to ChatServer's list of active threads
        } catch (IOException e) {
            e.printStackTrace();
            closeConnection();
        }
    }
```

ServerThread main loop: has multiple blocking function calls that read or write messages to the socket. These IO functions are the reason that this work is being done concurrently with threads:

```
    @Override
    public void run() {
        setupUser(); // blocking!!
        while (!socket.isClosed()) {
            try {
                String msg = in.readLine(); // blocking
                if (msg.equals(EXIT_CMD)) {
                    closeConnection();
                    userExit();
                } else if (msg.equals(ACTIVE_USERS_CMD)) {
                    listActiveUsers();
                } else {
                    userMessageAll(msg); // blocking
                }
            } catch (IOException e) {
                closeConnection();
            }
        }
    }
```

ChatClient

The ChatClient class manages input from and output to the terminal for each user, along with IO with the socket. It also makes use of threads to receive messages from the server while waiting for input from the user. Along with IO, it also manages validating the user name provided to avoid inputs like “Bye” or empty strings as names before sending the validated name to the server.

Main method: the call to listen() creates a new thread for receiving messages from the server and outputting to the terminal. setupUser() and send() are both run on the current thread, and manage validating and sending the user name, and sending messages from System.in to the socket, respectively:

```
public static void main(String[] args) {
    String ipAddress = args[0];
    int port = Integer.parseInt(args[1]);
    ChatClient client = new ChatClient(ipAddress, port);
    client.listen(); // blocking / creates new thread
    client.setupUser(); // blocking / gets input from sys in until valid name, no new thread
    client.send(); // blocking / get input from sys in, does not create new thread
}
```

setupUser() handles validating the input for the user name to make sure the user does not input whatever the quit message is set to ("Bye"), or an empty string – it sends the validated input to the server:

```
    private void setupUser() {
        String name;
        if (socket.isConnected()) {
            name = scanner.nextLine();
            while (name.equals("") || name.equals(EXIT_MSG)) {
                System.out.println("Invalid user name");
                name = scanner.nextLine();
            }
            try {
                out.write(name); // output validated name to server
                out.newLine();
                out.flush();
            } catch (IOException e) {
                closeConnection();
            }
        }
    }
```

Proof that it works!

Server startup message:

```
Server Socket created : ServerSocket[addr=0.0.0.0/0.0.0.0,localport=5000]
Server running... accepting clients
```

Entering a user name:

```
Welcome!  Enter user name:
foo
[22:25:55] Server: Welcome, foo
```

View from other terminal that hasn't entered user name yet:

```
Welcome!  Enter user name:
[22:25:55] Server: Welcome, foo
```

Chat with three clients:

```
Welcome!  Enter user name:
[22:25:55] Server: Welcome, foo
[22:28:57] Server: Welcome, bar
baz
[22:29:01] Server: Welcome, baz
hi lol wyd
[22:29:17] baz: hi lol wyd
```

```
Welcome!  Enter user name:
foo
[22:25:55] Server: Welcome, foo
[22:28:57] Server: Welcome, bar
[22:29:01] Server: Welcome, baz
[22:29:17] baz: hi lol wyd
lol nothin lol XD
[22:30:24] foo: lol nothin lol XD
```

Active users command:

```
[22:30:24] foo: lol nothin lol XD
AllUsers
Active Users at 22:30:53:
baz
bar
foo
```

Exit command:

```
[22:28:57] Server: Welcome, bar
[22:29:01] Server: Welcome, baz
[22:29:17] baz: hi lol wyd
[22:30:24] foo: lol nothin lol XD
Bye

Process finished with exit code 0
```

View from other terminal:

```
[22:29:01] Server: Welcome, baz
hi lol wyd
[22:29:17] baz: hi lol wyd
[22:30:24] foo: lol nothin lol XD
Server: Goodbye, bar
```

Validating the user name:

```
Welcome!  Enter user name:
Bye
Invalid user name

Invalid user name
```