A channel has a unique title and is started by a User; any other User can join the channel by adding themselves to the list of its members (issuing a *join* command). A channel is destroyed when all of its members have left, either by manually choosing to leave the channel or exiting the chat client.
The user communicates with the server through an open socket for each user.
Server structure:
The client sends commands to the server through the Connection class. The server has a thread for each client listening for the client's commands. The server has a queue for commands coming from clients.

public class Client

The client runs on the client's computer.

```
private User user;

// Create a new Client given a server and a socket.  Upon running,
Client creates an instance of Connection to connect to the server via
the socket in order to log in.
public Client(Server server, Socket socket);

// Run the Client
public void handleCommand();

// Handles clients to server requests.
public void handleCommand();
```

public class Connection *implements Runnable*

A new instance of this class gets created in a separate Thread whenever a user connects to the server. The Connection class processes all Client-Server communication on the Socket called socket, and delegates the execution of the commands passed by the client to the Server.

```
public Socket sock;
```

```
    private Server serv;
    private User u;

    // Constructor: Create a new Connection given a server, socket, and
    nickname
    public Connection(Server server, Socket socket, String nickname);

    // Process client commands as they arrive on Socket sock until the
    // "quit" command is received, or until the socket times out.
    public void run();

    // Handles clients to server requests by parsing the command and calling
the server method with appropriate parameters, and sends messages sent by the
server back to the client.
    public void handleCommand();
```

public class Channel

This class represents a channel in guichat, and stores a List of members (Users) and a List of Messages. Has functions to to process client commands such as "join-Channel" and "leave-Channel". An instance of Channel is created when at least one person joins the Channel, and then removed when it has no more members.

```
    private String name;
    private User owner;
    private List<User> users;
    private List<Message> messages;

    //Construct a Channel with the t
    public Channel(String channelName, User owner);
```

```
//Returns true if this Channel has User u as one of its current members;
//false otherwise
public boolean hasUser(User u);

//Add User u to this Channel's list of members
public void addUser(User u);

//Remove User u from this Channel's list of members
public void removeUser(User u);

//Send a message to the group by
//adding Message m to this Channel's List of Messages
public void sendMessage(Message m);

//Returns the users in this Channel
public List<User> getUserCount();

//Returns the number of users in this Channel.
public int getUsers();
```

public class Server

The chat server. Stores a HashMap linking Channel names to Channels, a HashMap of nicknames and their respective User classes, and a HashMap of Users to the Thread that processes them. Also has functions to carry out User commands.

```
private ServerSocket server;
private HashMap<int, User> userMap;
private HashMap<String, Channel> ChannelMap;

// Instantiate a server on the specified port.  Create Server fields of
```

```
// a ServerSocket, a HashMap<String, User> and a HashMap<String,
Channel>
public Server(int port);

// Listen for connections on the specified port; creates User and
// Threads to run
public void listen();

// Process a user "joining" a Channel, true if join successful
public boolean addUserToChannel(String nickName, String ChannelName);

// Process a user "leaving" a Channel, true if leave successful
public boolean removeUserFromChannel(int userID, String ChannelName);

// Process a user creating a Channel, true if creation successful
public boolean createChannel(int userID, String ChannelName);

// Return a Channel on the Server.
public Channel getChannel(String channelName);

// Return a User on the Server.
public User getUser(int userID);

//Remove a User that has disconnected from the Server
public void processUserDisconnect(int userID);
```

public class User

A class representing a user

```
        private Server serv;
        private int id;
        private String nickname;

        // Create a new User given a server, socket, and nickname
        public User(Server server, Socket socket, String nickname, int userID);
```

public class Message

This class represents the messages users send to each other in a chat which appear in the chat history.

```
        private User sender;
        private Channel Channel;
        private Date date;
        private String content;

        // Create a new message given an author, Channel, date, and message
content
        public Message(User sender, Channel Channel, Date date, String content);

        // Returns the contents of this Message
        public String getContent();

        // Returns the date that this message was sent
        public Date getDate();

        // Returns the sender of this message
        public User getSender();

        // Returns the Channel of this message
        public Channel getChannel();
```

```
    // Returns a chat version of the message: "{date} {sender}: {content}"
    public String toString();
```

Client-Server Protocol

Shared grammar:
WORD ::= [A-Za-z0-9_]+
SPACE := " "
NEWLINE := "\n" //Use as end of message character?
COMMA ::= ","
PERIOD := "."

NAME := [WORD]+
nickname = CHANNELNAME = NAME
ID := [0-9]+
LINE := [WORD SPACE COMMA PERIOD]*
TEXTLINE := [LINE]* NEWLINE

Client --> Server

protocol ::= command*
command ::= (login | logout | list-channels | list-users | user-channels | join | leave | post) NEWLINE
login := "login" nickname //logs in a user with that nickname
logout := "logout" //logs the current user out
list-channels := "list-channels" //list all open channels (channels with >0 users in them)
list-users := "list-users" //list all users currently online
user-channels := "user-channels" //list the channels the user is in

join := "join" CHANNELNAME

//have user join the channel with name CHANNELNAME

leave := "leave" CHANNELNAME?
//have user with name nickname leave the channel with name CHANNELNAME
//If no channel argument supplied, user leaves current channel

message := "message" CHANNELNAME TEXTLINE
// have user with name nickname post a message comprising of TEXTLINE in the channel with name CHANNELNAME

Server --> Client

protocol ::= command*
command := success | failure | error | reply-list-channels | reply-list-users

success := "success"
failure := "failure" TEXTLINE* // TODO: have TEXTLINE contain helpful info about failure
error := "error" TEXTLINE // When server throws Exceptions
reply-list-channels := "reply-list-channels" CHANNELNAME*
//info message sent in response to a list-channels or user-channels request
reply-list-users := "reply-list-users" nickname*
//info message sent in response to a list-users request

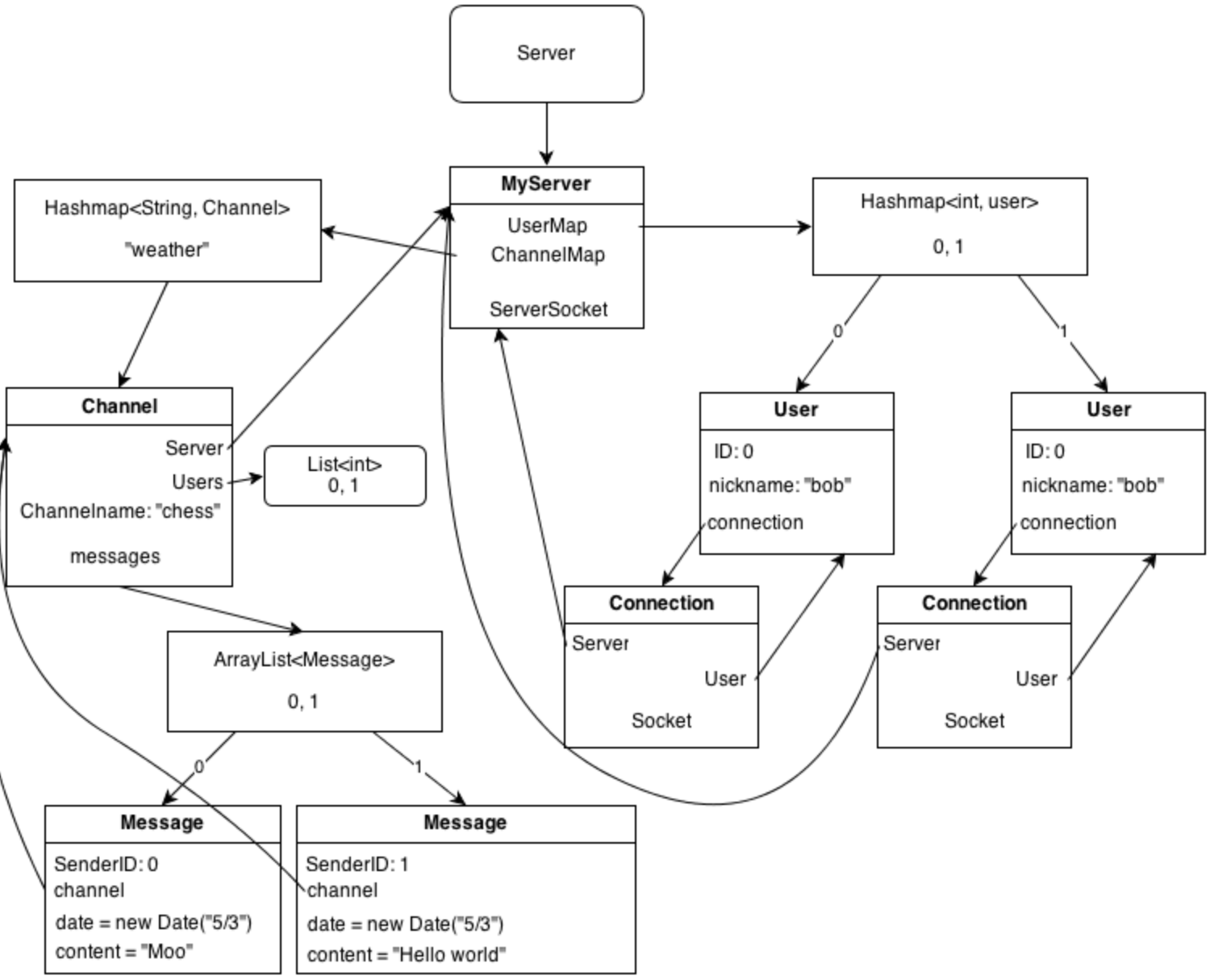
Testing strategy

The main testing strategy is to start a dummy server listening on a port, and then simulate a client connecting to this dummy server and processing different actions (creating a Channel, quitting a Channel, closing the client... etc) and verifying that both the response of the server is acceptable and that the Channels and messages specified in the test cases are actually created and are readable on the client.

Also we will have several test cases with multiple clients where one of the clients will try to read and respond to a conversation sent by the first client - we can do this by verifying the input & output sockets for different users and checking that the message order is preserved.

In general, we want to test all commands in our Client-Server protocol, and ensure that the messages sent back to the client from the server are correct.

Snapshot Diagram


Server
MyServer
UserMap
ChannelMap
ServerSocket
Hashmap<String, Channel>
"weather"
Hashmap<int, user>
0, 1
0
1
Channel
Server
Users
Channelname: "chess"
messages
List<int>
0, 1
User
ID: 0
nickname: "bob"
connection
User
ID: 0
nickname: "bob"
connection
Connection
Server
User
Socket
Connection
Server
User
Socket
ArrayList<Message>
0, 1
0
1
Message
SenderID: 0
channel
date = new Date("5/3")
content = "Moo"
Message
SenderID: 1
channel
date = new Date("5/3")
content = "Hello world"