# Team Contract

## Goals

- What are the goals of the team?
    - To learn how to make an integral part of the evolution of the internet, and have fun while doing it.
- What are your personal goals for this assignment?
    - Chase - Learn how to program a chat client really quickly and efficiently
    - Sebastian - Learn how to make a sweet UI in Java
    - Thomas - Learn how to deal with server/user protocols efficiently
- What kind of obstacles might you encounter in reaching your goals?
    - Coordinating times for meeting each other; working with svn (we might try git)
- What happens if all of you decide you want to get an A grade, but because of time constraints, one person decides that a B will be acceptable?
    - We will try and talk to them. If that doesn't work, the two who want an A will sadly have to pick up the slack.
- Is it acceptable for one or two team members to do more work than the others in order to get the team an A?
    - If it comes down to that, then yes.

## Meeting Norms

- Do you have a preference for when meetings will be held? Do you have a preference for where they should be held?
    - Student center; Tuesdays/Thursdays around 4
- How will you use the in class time?
    - To meet and hack.
- How often do you think the team will need to meet outside of class? How long do you anticipate meetings will be?
    - Once or twice this week. Probably around an hour.
- Will it be okay for team members to eat during meetings?
    - Absolutely
- How will you record and distribute the minutes and action lists produced by each meeting?
    - Google docs or email or source code control

## Work Norms

- How much time per week do you anticipate it will take to make the project successful?
    - Probably 15 hours each per week.
- How will work be distributed?
    - By consensus
- How will deadlines be set?
    - By consensus. We should probably do everything ahead of schedule. General goal is to frontload work and finish as soon as possible, without compromising quality of work.
- How will you decide who should do which tasks?
    - Whoever wants what; otherwise straw pulling.
- Where will you record who is responsible for which tasks?
    - Google docs.
- What will happen if someone does not follow through on a commitment (e.g., missing a deadline, not showing up to meetings)?
    - We will talk to them, and work together to make sure the issue is resolved
- How will the work be reviewed?
    - Github code review

- What happens if people have different opinions on the quality of the work?
    - We will talk about it. We will learn from each other.
- What will you do if one or more team members are not doing their share of the work?
    - Ask them to do more; bribe them with cookies.
- How will you deal with different work habits of individual team members (e.g., some people like to get assignments done as early as possible; others like to work under the pressure of a deadline)?
    - If the work is done by the deadline we set, then it doesn't matter when it was done.

## Decision Making

- Do you need consensus (100% approval of all team members) before making a decision?
    - No; 2/3
- What will you do if one of you fixates on a particular idea?
    - Talk about it and hopefully come to some compromise.

Signed on May 1:

Chase Lambert
Sebastian Leon
Thomas Schultz

## Conversation Design

**Conversation design.**
Define a precise **notion** of *conversation* in your IM system.
A conversation is between two people. There are three states:
Ended (this means that the conversation doesn't exist)
Waiting - waiting for another person to accept/reject
Running - two people can talk to each other

**State Diagram**
Name the Java classes you will create to implementing conversations
**Conversation**
Holds the users connected and messages. Just a data structure; so getters and setters.
**Messages**
Holds text, the sender's username, the recipient's username, and time stamp; so getters and setters.
**ChatClientHandler**
The server that holds all the client connection. Waits on new connections, adds them to the list of connections and goes through this loop.

```
ArrayList<ClientThread> clientThreads;
public void handleClients() {
        // goes through a loop waiting for clients to connect
}
```

**ChatServer**
Listens to the client threads for any new information. We are doing this to prevent any concurrency issues. By handles, we mean that it saves the message and then propagates the message to the proper users.

```
public void relayMsg(Message msg) {
        // called by a client thread and relays the message to the appropriate thread
}

public void disconnect(String username) {
        // tells all other clients to see that the username is disconnected
}

public void connect(String username) {
        // tells each client that a new user has logged in
}
```

**ClientThread**
Handles each client; holds all the messages sent by the client. Sends an event to the server when a new message is made.

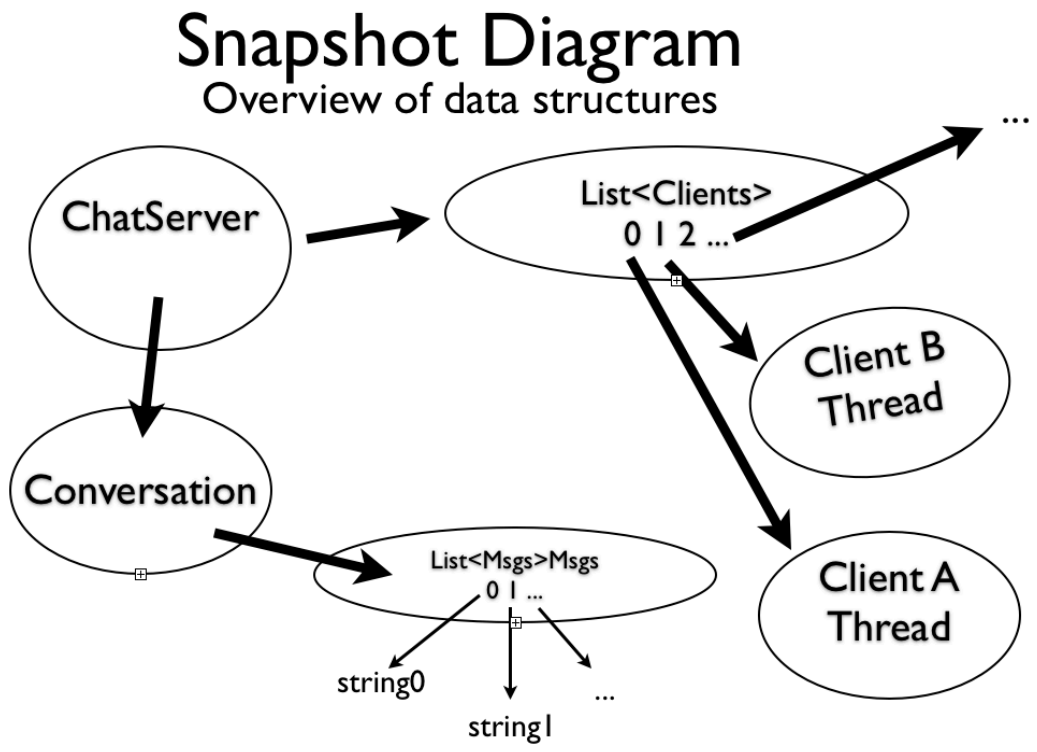
```
private void handleConnection() {
        //  calls handleRequest whenever a new message is sent by this client; waits on each new
message
}
```

```
private void handleRequest(Message msg) {
        // sends the message to the server to be relayed
}

public void receiveMessage(Message msg) {
        // called by the server; tells the client that this new message has been received
}
```

**ClientGUI**

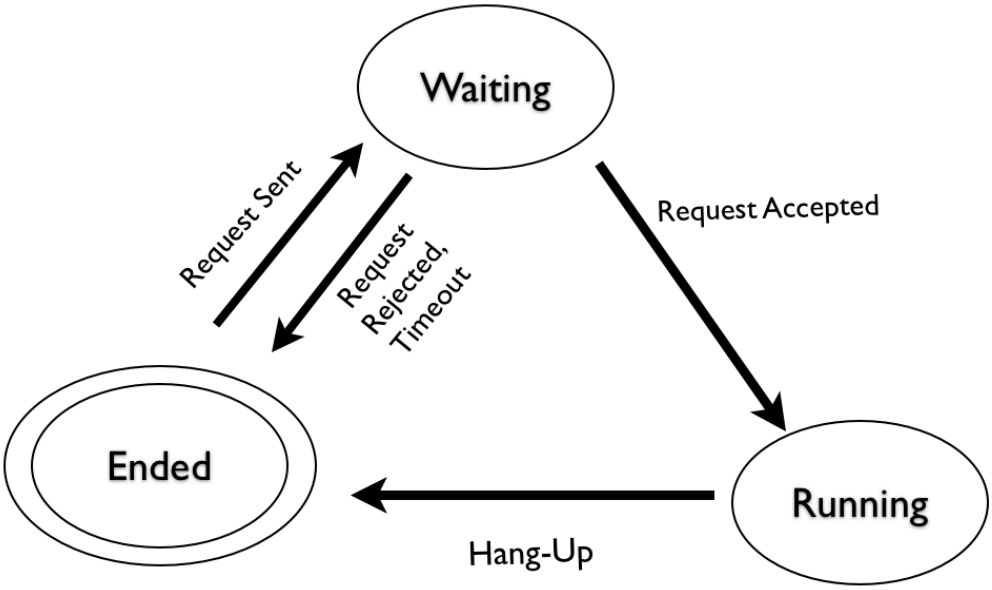
Creates the GUI; handles all the events on the GUI; sends and receives messages to and from the server.

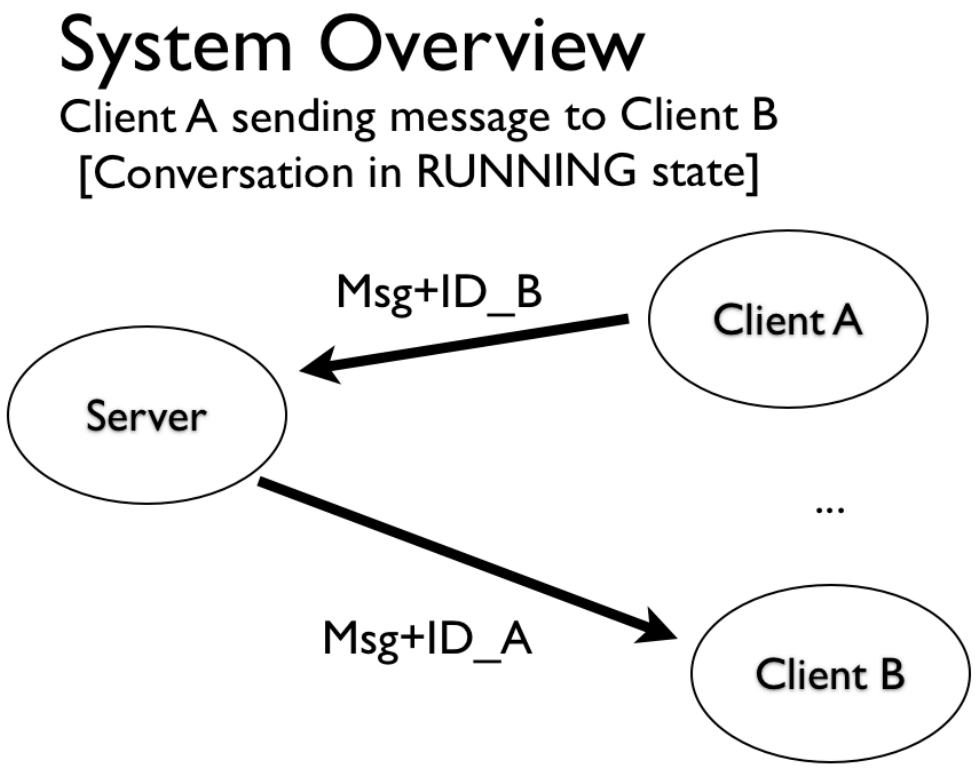
```
public void sendMsg(Message msg) {
        // sends a message to the server
}
public void sendChat(Message msg) {
        // calls sendMsg to send this message
}

public void loginAs(String username) {
        // calls sendMsg to send a login command to the server as this name
}

public void receiveMsg(Message msg) {
        // called when the server sends a message to the client
}
```



# Conversation Design

2 way communication between users






**Protocols:**

**Client/Server protocol:**

User-to-Server Message Protocol
MESSAGE:== (CONNECT | MSG)
MSG :== MSG_TEXT SPACE TO_NAME SPACE .+ // Content that is being sent from one user to another
MSG_TEXT :== “MSG”
CONNECT :== CONNECT_TEXT SPACE \w+ // Unique username for every user, defined when user enters system
CONNECT_TEXT :== “CONNECT”
SPACE :== “ “ // just a space

**Server/Client protocol:**

Server-to-User Message Protocol
MESSAGE:== MSG_TEXT SPACE FROM_NAME SPACE MSG
MSG_TEXT :== “MSG”
SPACE :== “ “ // just a space
MSG :== “MSG .+” // Content that is being sent from one user to another
FROM_NAME :== “\w+” // Unique ID for every user, defined when user enters system