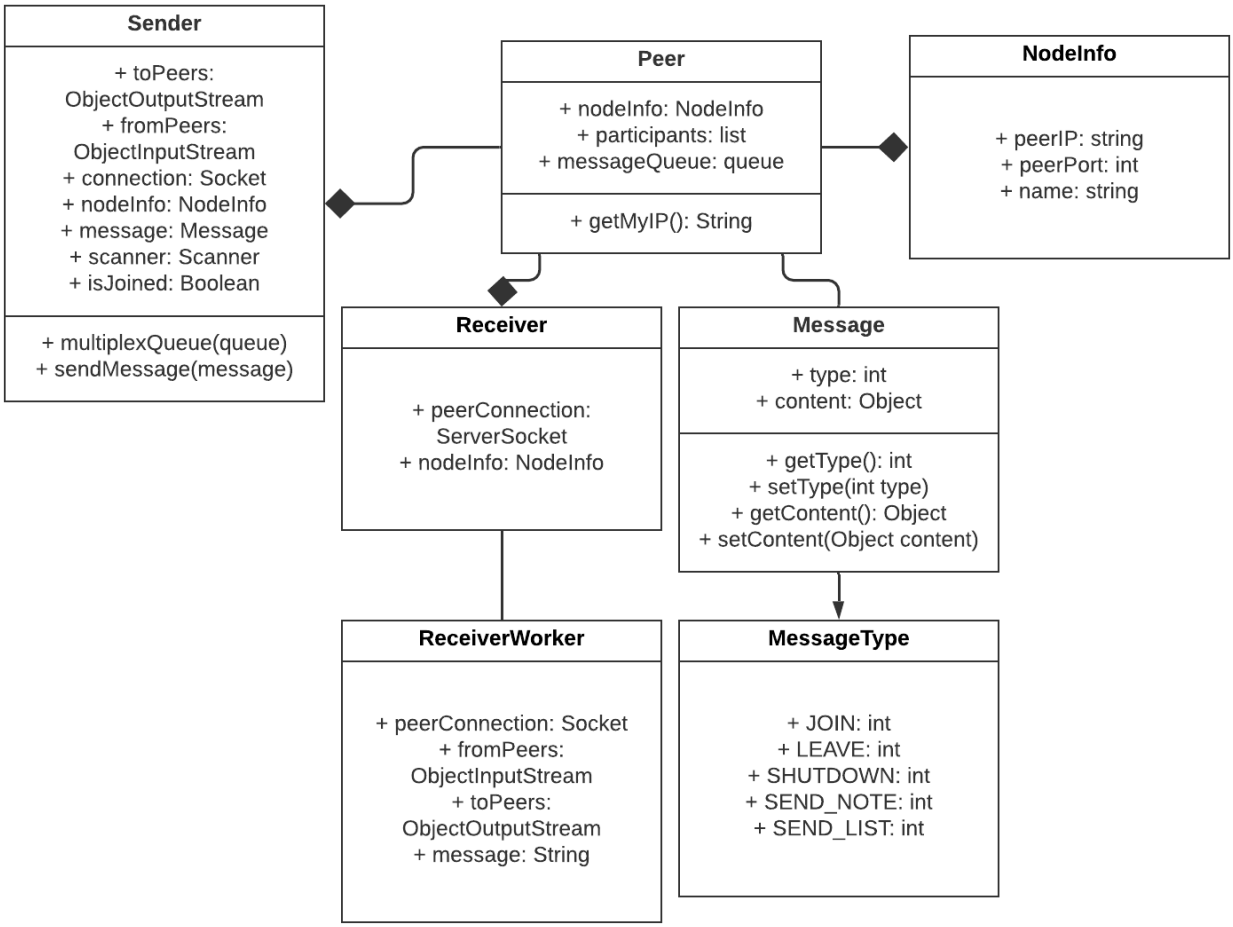
Distributed Chat, Part 2 - Design

Joe Domabyl V
Daniel Rydberg
Nick Nannen

Topology Description

There are a number of users (called peers), any of whom can join the chat at any time using a JOIN request. Each user is both a client and a server, allowing them to directly connect to one another without using a central server as a middleman, making this a true peer-to-peer chat. We previously considered using a “master” peer to multiplex messages to all connected users, but this would be a central server in fact if not in name, so we had to scrap this idea. Each peer will maintain records of who is connected via NodeInfo objects within an ArrayList. From there, peers will be able to interact with each other via a series of different message types, described in detail under “Messaging Protocol”.

Class Diagrams



The system will be designed with 7 classes. The primary class, **Peer** is associated with the **NodeInfo** class, the **Sender** class, the **Receiver** class, and the **Message** class. **NodeInfo** will contain all information that defines a single peer, their IP, their port, and their human-readable name. The **Sender** will be responsible for connecting to other receiver threads and multiplexing any messages within the queue. The **Receiver** will listen for connections on the peers port and accept any messages from other senders via the **ReceiverWorker** class. The **ReceiverWorker** will be responsible for handling incoming messages. The **Message** class will define different **MessageType**s, readable by peers and handled by connected classes.

Messaging Protocol

Our messaging protocol consists of four distinct types; JOIN, LEAVE, SHUTDOWN, and NOTE. Many of these types work much as they did in the first project, but there are also some differences.

The JOIN message type connects a user to one other user assuming that they know the port and IP of said user. The user that is being connected to then sends the new user the ArrayList consisting of members of the chat using a definition called SEND_LIST. The new user then updates the ArrayList it has been sent with its own NodeInfo and sends the new ArrayList to all the other users using the same SEND_LIST definition. The other users then update their own ArrayList with the new user information.

The LEAVE message simply disconnects the user who sent it from the chatroom. This does not shut down the program for the user, just simply sends them back to the starting screen where they can then join another chatroom.

The SHUTDOWN message type shuts the entire system down, disconnecting all users and terminating the program. This can be initiated by any user at any time to shutdown the chatroom.

Finally, the NOTE message type simply sends a text message to the chatroom. This is accomplished by putting the message that a user is going to send into their own message queue and then sending the messages to each user in the user ArrayList.