**Bus - implemented as a doubly-linked list .**

* At any given time, there is at least one active "participant" in the chat application. This is how e.g. the chat application starts out.
* Any non-participating node interested in chatting with the already existing participanting nodes can contact any one of them with a JOIN request. Consequently, any participant can handle a join request. We assume that a non-participating node "just knows" the IP address and port number of one other node that is a participant.
* Messages sent out by one participant are generally broadcast to all other, current participants and contain a logical name of the sender, apart from the note itself.
* A participant may leave the chat at any time.
* Any one client may shutdown the whole chat.
* There is a command-line interface in place that allows a user to control a chat node. This is supposed to be modeled after the central server chat application.
* Communication channels may be kept open or may be closed after each interaction. I suggest to do the latter.
* To keep the system simple, we assume that new participants don't see old messages, i.e. there is no concept of "history".
* There is no assumption about a limit on the number of participants.
* Whatever you do, this is (strictly) a peer-to-peer application, there is no central server allowed. Note, however, that each of the participants will need to have a server component.

**In order to trigger all above activities/states, the client needs to provide a primitive command-line user interface for the user to use. The interface will recognise the following "commands":**

1. Description of bus Topology: Describe the actions involved, each one in its own section, clearly deliniated:
2. Joining a chat,
3. Leaving a chat,
4. Sending/receiving messages and
5. Shutting down the chat peer
6. Shutting down the chat as a whole